VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KOMPONENTY UŽIVATELSKÉHO ROZHRANÍ V PHP

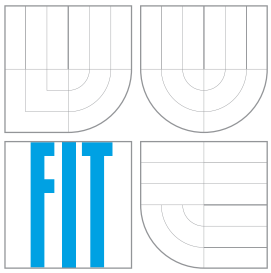
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VOJTĚCH KULIŠŤÁK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KOMPONENTY UŽIVATELSKÉHO ROZHRANÍ V PHP

USER INTERFACE COMPONENTS IN PHP

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VOJTĚCH KULIŠŤÁK
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK BURGET, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Primárním cílem této práce je návrh a implementace komponent uživatelského rozhraní v jazyce PHP. Dále jsou zde nastíněny nejpoužívanější jazyky pro tvorbu webových aplikací. Největší pozornost je věnována jazyku Javascript a PHP. Implementace komponent byla provedena s použitím frameworku Nette spolu s preprocesorem XTemp. Kromě zmíněného frameworku Nette s preprocesorem XTemp jsou v práci okrajově popsány možnosti tvorby komponent s použitím aplikačních rámců Zend Framework a CodeIgniter.

Abstract

The primary goal of this bachelor's thesis is design and implementation of user interface components in PHP. Furthermore, are discussed the most widely used languages for creating web applications. The greatest attention is paid to JavaScript and PHP. Implementation of the components was made using Nette framework together with preprocessor XTemp. In addition to the listed Nette framework and XTemp preprocessor this thesis marginally describes possibilities for creating components using frameworks Zend and CodeIgniter.

Klíčová slova

framework, komponenty, Nette framework, XTemp, PHP, webová aplikace, uživatelské rozhraní

Keywords

framework, components, Nette framework, XTemp, PHP, web application, user interface

Citace

Vojtěch Kulišťák: Komponenty uživatelského rozhraní v PHP, bakalářská práce, Brno, FIT VUT v Brně, 2015

Komponenty uživatelského rozhraní v PHP

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Kulišťák
19. května 2015

Poděkování

Rád bych poděkoval svému vedoucímu bakalářské práce panu Ing. Radkovi Burgetovi, Ph.D. za poskytnuté cenné rady a také za rychlé a vstřícné domluvy konzultačních schůzek.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Vytváření webových aplikací, jejich vývoj a udržování v podobě zaručující konkurenceschopnost, je doprovázeno velkým množstvím problémů a úskalí, kterým je potřeba věnovat pozornost už při samotném návrhu webové aplikace. V dnešní době existuje velké množství různých standardů, konvencí, návrhových vzorů, technik nebo nástrojů ulehčujících řešení těchto problémů. Jejich popis a doporučené použití je probíráno v mnoha článcích či odborných publikacích (příkladem může být publikace zabývající se procesem vývoje webových aplikací *"Open Source - vývoj webových aplikací"* [16], nebo kniha s názvem *"Creating a Website: The Missing Manual"* [20], jejímž autorem je Matthew MacDonald). Jedním z důležitých faktorů kvalitní webové aplikace je přehlednost a logická strukturalizace zdrojových kódů. Té lze docílit rozdělením aplikace do samostatně fungujících komponent, představujících základní kámen znovupoužitelnosti zdrojového kódu. Součástí této práce je takovéto komponenty navrhnout a implementovat.

Primárním cílem této práce je návrh a implementace komponent uživatelského rozhraní v jazyce PHP. V teoretické části práce je popsána tvorba webové aplikace se zaměřením na existující jazyky a nástroje pro implementaci jak serverové tak klientské strany aplikace. Dále jsou zde popsány aplikační rámce založené na platformě PHP. Aplikační rámce jsou rozebrány z hlediska jejich možnosti tvorby znovupoužitelných komponent. Největší pozornost je přitom věnována frameworku Nette, který byl použit pro implementaci navržených komponent. Kromě frameworku Nette je k implementaci navržených komponent používán preprocesor XTemp. Preprocesor XTemp je popsán v samostatné kapitole a největší prostor je zde věnován principu zpracování XHTML souboru, definujícího pohled aplikace. Druhá část této práce se zabývá popisem implementace a použitím navržených komponent. V samotném závěru je popsána aplikace demonstrující funkčnost navržených komponent, způsob testování a také možnosti rozšíření.

Kapitola 2

Vymezení základních pojmů

V této kapitole budou vysvětleny některé technologie a pojmy týkající se problematiky tvorby webových aplikací, jenž se budou vyskytovat v následujícím textu. Jedná se pouze o výčet pojmů se stručným popisem, beze snahy jakéhokoliv členění či strukturalizace textu.

HTML

HTML[1] je hypertextový značkovací interpretovaný jazyk, používaný pro tvorbu webových stránek. Tento jazyk vychází z univerzálního jazyka SGML (*Standard Generalized Markup Language*) a jeho hlavním účelem je definice podoby (textu a dalších objektů) webového dokumentu. [25]

XHTML

XHTML[2] je hypertextový značkovací jazyk, používaný pro tvorbu webových stránek. Tento jazyk byl dne 26. ledna 2000 doporučen konsorciem W3C a revidován 1.srpna 2002. Jazyk XHTML vychází z obecného standardu pro výměnu dat XML (*Extensible Markup Language*).[30]

CSS

CSS[3] je jazyk pro popis reprezentace a způsobu zobrazení jednotlivých elementů pro jazyky HTML, XHTML případně XML. Tento jazyk vznikl v roce 1996, z příčiny nekonzistentní podpory webových prohlížečů bylo ale jeho použití po dlouhou dobu značně omezeno. Definice zobrazení elementů pomocí kaskádových stylů je udržována mimo strukturu a obsah dokumentu, z čehož plyne větší přehlednost, dostupnost a rychlejší načítání stránek.[10]

API

API[4], neboli rozhraní pro programování aplikací, je sada rutin, protokolů a nástrojů pro tvorbu softwarových aplikací.[2]

[1] HyperText Markup Language
[2] Extensible Hypertext Markup Language
[3] Cascading Style Sheets (do češtiny přeloženo jako kaskádové styly)
[4] Application Program Interface

Dependency Injection

Dependency injection (DI) je návrhový vzor sloužící pro předávání závislostí mezi jednotlivými komponentami. Cílem tohoto návrhového vzoru je získání čitelného a znovupoužitelného kódu. Vytváření závislostí a jejich následné předávání se implementuje v objektu zvaném *DI kontejner*. Komponenta tak závislost získá od tohoto objektu (např. pomocí parametru v konstruktoru třídy) a nemusí si ji získávat sama. [14]

Architektura MVC

Architektura MVC (Model, View, Controller) je návrhový vzor, definující rozdělení aplikace do 3 logických částí.

1. Model - Zajišťuje přístup k datům určitého datového úložiště (např. databáze) a manipulaci s těmito daty.
2. View (Pohled) - Představuje výslednou reprezentaci dat daných modelem v podobě vyhovující koncovému uživateli.
3. Controller (Řadič) - Zprostředkovává reakce uživatele a reaguje na ně upravením modelu nebo pohledu.

Tímto rozdělením aplikace získávají na přehlednosti a jejich zdrojové kódy se lépe udržují a rozšiřují.

REST

REST[5] je architektonický styl popisující návrh API pro bezstavové služby. Architekturu REST ve své disertační práci *Architectural styles and the design of network-based software architectures* popsal v roce 2000 americký počítačový vědec Roye T. Fielding [11]. Jedná se o architekturu orientovanou na zdroje (tzv. ROA[6]).

JSON

JSON[7] je textový formát, určený pro výměnu strukturovaných dat. Je založen na podmnožině jazyka Javascript (ECMAScript). JSON data reprezentuje jako univerzální datové struktury, které podporují prakticky všechny moderní programovací jazyky.[9]

[5] Representational state transfer
[6] resource-oriented architecture
[7] JavaScript Object Notation

Kapitola 3

Tvorba webové aplikace

Jako webovou aplikaci můžeme označit každou aplikaci, která je uživatelům poskytovaná z webového serveru přes počítačovou síť internet, typicky prostřednictvím webového prohlížeče. V této kapitole budou popsány techniky a jazyky používané k vytváření interaktivních webových aplikací. Bude nastíněna tvorba serverové části aplikace, se zaměřením na jazyk PHP a aplikační rámce vyvíjené pro podporu tohoto jazyka. Vyjma jazyka PHP, bude u kapitoly zabývající se problematikou vývoje serverové části aplikace popsáno prostředí Node.js, přinášející na stranu serveru skriptovací jazyk Javascript. Dále zde budou probrány možnosti implementace klientské strany aplikace, kde hlavní pozornost bude věnována jazyku Javascript, jeho nedostatkům a možnostem, jak lze tyto nedostatky alespoň částečně eliminovat. V závěru kapitoly bude popsán soubor technologií AJAX.

3.1 Serverová část webové aplikace

Serverová část aplikace je v nejčastějších případech používána k poskytnutí zdrojů dat (např. z databáze), které v předem specifikované formě poskytuje klientské straně. Z důvodu výkonnějších strojů (oproti klientské straně) použitých u serverové části webové aplikace, může být tato část taktéž použita k implementaci většiny aplikační logiky. V tomto případě je minimalistický klient použit pouze k prezentaci dat poskytovaných serverem. V dnešní době je pro implementaci serverové části webové aplikace nejvíce používán programovací jazyk PHP. Podle statistiky uváděné webovým serverem w3techs.com, zaznamenané ke dni 19.4. 2015, je jeho použití v rámci programovacích jazyků implementujících serverovou část aplikace, zastoupeno z celých 82% [29]. Druhou příčku ve statistice nejpoužívanějších technologií pro implementaci serverové části aplikace zastává webový framework ASP.NET. Tento framework obsahuje sadu knihoven obsahujících hotová řešení, která se týkají problematiky tvorby webových aplikací. Webové aplikace vyvíjené pomocí tohoto frameworku jsou psány v jazyce *C#*. Statistika taktéž uvádí, že v poslední době narůstají webové servery, jejichž implementace je závislá na programovacích jazycích jako je *ColdFusion*, nebo *Javascript*. Tento trend plyne z faktu neustále rostoucích požadavků na rychlost interaktivních webových aplikací.

3.1.1 PHP

PHP je skriptovací multiplatformní programovací jazyk, jehož zkratka původně pocházela ze slovního spojení *"Personal Home Page"*. Dnes je reprezentován názvem *"PHP: Hypertextový preprocesor"* a představuje vůbec nejpoužívanější jazyk pro tvorbu webových aplikací

a dynamických internetových stránek. Počátky jazyka PHP se datují do roku 1994, kdy jej vytvořil dánsko-kanadský programátor Rasmus Lerdorf [17]. Tento programovací jazyk je interpretovaný na serverové straně webové aplikace. Uživateli jsou přenášeny pouze výsledky jeho činnosti. Mezi výhody tohoto jazyka patří jednoduchá syntaxe velice podobná jazyku *C*, snadná implementace komunikace s databázemi, rozsáhlý soubor funkcí, nebo podpora široké řady souvisejících technologií a formátů.

3.1.2 Node.js

Systém Node.js představuje výkonné událostmi řízené vývojové prostředí pro vývoj serverové strany interaktivních webových aplikací. Prostředí Node.js začal vytvářet v roce 2009 Ryan Dahl a je založeno na skriptovacím jazyku Javascript, interpretovaném pomocí tzv. *V8 Javascript Engine* [24]. *V8 Javascript Engine* je interpret vytvořený firmou Google, použitý poprvé ve webovém prohlížeči Google Chrome [22]. Tento interpret se vyznačuje především rychlostí interpretace javascriptového kódu, kdy jej před samotným spuštěním překládá do strojového kódu. Využití systému Node.js na straně serveru přináší několik pozitivních aspektů. Za zmínění stojí především možnost sdílení kódu mezi klientem a serverem, asynchronní I/O operace, podpora pro real-time[1] aplikace, nebo výběr z velkého množství již vytvořených modulů. Další informace o tomto vývojovém prostředí lze získat např. z oficiální webové dokumentace dostupné na adrese *http://www.nodejs.org/documentation/* nebo z knihy *Practical Node.js* [22], případně *Node.js for PHP Developers* [15].

3.2 Klientská část webové aplikace

Jako nejpoužívanější programovací jazyk současnosti se pro implementaci klientské strany webové aplikace používá programovací jazyk Javascript. Statistika webu *w3techs.com* [27], jenž se tímto tématem zabývá uvádí, že se jeho použití (ke dni 16. dubna 2015) týká 89% z celkového podílu webů. Jako další příklad programovacího jazyka klientské strany webové aplikace může být uveden programovací jazyk *ActionScript*[2], jenž slouží k rozvinutí všech možností interaktivních animací a vývoji aplikací ve vektorovém programu *Flash*. Jazyk Javascript se od svého vzniku hodně vyvinul a neustále se vyvíjí. V této podkapitole, budou kromě jazyka samotného, popsány některé z nejpoužívanějších javascriptových knihoven, nebo aplikačních rámců.

3.2.1 Javascript

Multiplatformní, objektově orientovaný skriptovací jazyk Javascript byl roku 1995 vytvořen tehdejší firmou Netscape a jeho autorem je americký programátor Brendan Eich [1]. První prototyp tohoto jazyka nesl název *Mocha*, později *LiveWire* a *LiveSript*. Název Javascript získal z marketingových důvodů podle jazyka Java, jenž v té době zažíval velký rozmach [18]. Javascript je jazyk interpretovaný na straně klienta. Tento fakt znemožňuje použití Javascriptu k centrálnímu uchovávání údajů týkajících se provozu webových stránek. Z důvodu bezpečnosti je navíc jazyk Javascript záměrně ochuzen o funkce pro práci se soubory. Je pravdou, že tyto omezení značně limitují okruh využití Javascriptu, jako majoritního

[1] Real-time aplikace jsou aplikace, vytvářené takovým způsobem, aby poskytovali reakce (např. krátké zprávy) na různé podněty (např. od uživatele) v co nejkratším čase.

[2] Další informace o tomto programovacím jazyce jsou dostupné na adrese *http://www.actionscript.org/*.. Jako další zdroj informací lze použít knihu *ActionScript 3.0 Bible* [6]

jazyka pro vytvoření plnohodnotných webových aplikací [18]. Na základě této skutečnosti byl jazyk Javascript považován především jako podpůrný jazyk, bez vysoce optimalizované implementace a jeho primárním využitím bylo:

- Ovládání interaktivních prvků, na základě vzniku určité události (př. událost mouse-over,mouseout,change,click...).
- Validace formulářů na straně klienta, před samotným odesláním formuláře na server (tvorba tzv. *chytrých formulářů*).
- Tvorba vyskakovacích oken.
- Využití údajů o aktuálním datu, času a jejich zobrazení.
- Tvorba různých animací a efektů obrázků.

Výše zmíněné využití jazyka převyšuje i v současné době, přesto je jazyk Javascript mnohými odborníky považován za velice silný nástroj a jeho využití se začíná rozšiřovat i do oblastí jako je zpracování serverové části aplikace (Node.js). Obecně se jazyk Javascript neustále vyvíjí a pro jeho lepší použitelnost vzniká velké množství různých aplikačních rámců a knihoven. Jedny z těch nejrozšířenějších a nejpoužívanějších budou nastíněny v následující podkapitole.

3.2.2 Nedostakty Javascriptu a jejich řešení

Aby nedošlo k mylnému dojmu, je nejprve nutno zmínit, že pojem "nedostatky" javascriptu není úplně přesný. Jedná se spíše o jakési omezení jazyka, s kterými je na základě ideologie jeho použití, způsobu interpretování a také faktu rozdílné interpretace v jednotlivých webových prohlížečích nutno počítat. Některé "nedostatky" javascriptu byly nastíněny již v předchozí podkapitole. V této podkapitole bude cílem představit některé frameworky, knihovny a také implementace založené na Javasciptu, jenž se snaží tyto "nedostatky" řešit a patřičným způsobem eliminovat.

Javascriptové knihovny

S přehledem nejpoužívanější javascriptovou knihovnou dnešní doby je knihovna JQuery. Podle statistiky webu W3Techs.com [28] ji ke dni 16. dubna 2015 používá 63.4% veškerých webů. Statiska navíc uvádí, že 33.3% webů ke svému chodu nepoužívají knihovnu vůbec žádnou, což z JQuery činí knihovnu, jejíž používaní je mezi javascriptovými knihovnami zastoupeno z celých 95%. Knihovna JQuery byla vydána roku 2006 Johnem Resigem a je šířena jako svobodný a otevřený software pod licencí MIT[3] [7]. Knihovna klade důraz na implementaci tzv. nevtíraného Javascriptu, kdy se jednotlivé události iniciující konkrétní javascriptové funkce definují mimo strukturu HTML. JQuery k této definici používá selektory, jejichž počtem rozšiřuje selektory používané v CSS. Další informace o knihovně JQuery jsou dostupné online na adrese *http://www.jquery.com*. Pro podrobnější studium může posloužit obsáhlá publikace *Learning jQuery* [7], na jejímž zpracování spolupracoval i autor knihovny John Resig.

[3] `MIT` (někdy označovaná jako licence *X11*, nebo *MIT X* licence) je svobodná licence vytvořená na Massachusettském technologickém institutu.

Jako příklad další knihovny lze uvést knihovnu *Modernizr*[4]. Tato knihovna najde uplatnění hlavně při nástupu nových technologií HTML5 a CSS3, kdy se s její pomocí dokáže detekovat podpora určitých vlastností zmíněných technologií u jednotlivých webových prohlížečů. Na základě zjištění této podpory lze poté určit patřičné styly z CSS stylopisu.

Existuje i velké množství dalších javascriptových knihoven, příkladem toho jsou knihovny *Bootstrap*, *MooTools*, *Prototype*, *Google Closure* nebo *ASP.NET Ajax*. Všechny tyto knihovny obohacují jazyk Javascript o další užitečnou funkcionalitu a jejich použití se při vývoji interaktivních webových aplikací velmi často kombinuje.

Javascriptové frameworky

Stejně jako javascriptových knihoven, existuje i velké množství javascriptových aplikačních rámců, neboli javascriptových frameworků. Za zmínění stojí například javascriptový framework *AngularJS*, jenž je vyvíjen firmou Google a představen byl roku 2009. Mezi základní přednosti *AngularJS* patří tzv. *Two Way Data-Binding*, řešící synchronizaci stavů mezi modelem a pohledem (v terminologii architektury MVC). Jako další pozitivum může být uvedena znovupoužitelnost komponent[5], testovatelnost nebo implementace tzv. *Dependency Injection*. AngularJS lze stáhnout z adresy *http://www.angularjs.org/*, kde je k dispozici dokumentace tohoto frameworku s kvalitně zpracovaným výukovým tutoriálem. Jako zdroj informací v podobě tiskovin může posloužit kniha *AngularJS Services* [19] nebo *Beginning AngularJS* [13] od Andrew Granta.

Dalším hojně používaným javascriptovým framworkem je *Ember.js*[6]. Tento framework byl představen roku 2011 a je šířen jako open-source[7] projekt pod licencí MIT. Ember.js představuje framework pro snadnou tvorbu vlastních komponent a šablon. Kromě rozdělení aplikace dle klasické architektury MVC *Ember.js* používá i tzv. routy, objekty reprezentující aktuální stav aplikace. Tyto objekty své uplatnění najdou například v kombinaci s tzv. *REST* rozhraním.

Javascriptových frameworků je v dnešní době poměrně velké množství a jejich rostoucí počet, dává najevo, že jazyk Javascript a vývoj interaktivních webových aplikací obecně, prochází neustálým vývojem.

Nové jazyky založené na Javascriptu

Alternativou pro řešení javascriptových "nedostatků" mohou být nové jazyky, které se do Javascriptu kompilují. Příkladem takového jazyka je např. jazyk *Dart*, vyvíjený firmou Google. Tento jazyk, syntaxí podobný jazyku *C*, byl představen roku 2011. Mezi jeho největší přednost patří zejména řešení nekompatibility webových prohlížečů. Kód napsaný v jazyce *Dart* se totiž pomocí kompilátoru *dart2js* překládá do jazyka Javascript a všechny rozdíly jednotlivých verzí prohlížečů řeší automaticky [3]. Jako další přednost lze uvést implementaci kvalitních standardních knihoven, znovupoužitelnost kódu nebo tzv. nepovinné statické

[4] Více informací na adrese *http://modernizr.com/*.

[5] Termínem *komponenta*, uváděným v tomto textu, bude myšlen softwarový celek, zajišťující funkcionalitu určité problematiky, jenž je možné opakovaně používat.

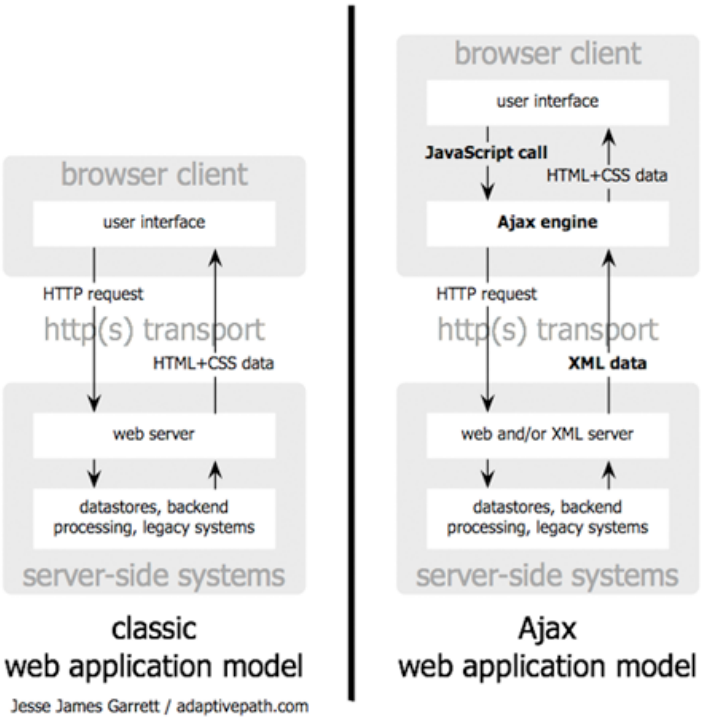
[6] Více informací na adrese *http://emberjs.com/*.

[7] Jedná se o tzv. *otevřený software*. Tento termín označuje dostupnost kódu z hlediska technického i legálního, kdy za určitých podmínek, umožňuje daný kód prohlížet a upravovat.

typování, které je vnímáno spíše jako anotace, nemající vliv na běh programu. Další velkou předností je možnost použití existujícího javascriptového kódu, což umožňuje používat knihovny, nebo funkce napsané dávno před vznikem tohoto jazyka. Pro studium jazyka Dart lze použít kniha *Web Programming with Dart*[3].

3.3 AJAX

AJAX (Asynchronous JavaScript and XML) je obecné označení pro soubor technologií umožňující asynchronní komunikaci klientské části webové aplikace se serverovou částí aplikace. Pojem AJAX poprvé použil Jesse James Garrett ve své práci, kterou publikoval v únoru roku 2005 [1]. V této práci Garrett mimo jiné popisuje, jak se postupně redukují rozdíly mezi klasickými klientskými a webovými aplikacemi. Mezi první aplikace používající AJAX patří aplikace od firmy Google, který AJAX zahrnul do projektů jako *Google Maps*, nebo *Gmail*. Asynchronní komunikace je při použití technologií AJAX iniciována klientem, kdy je ve formě tzv. XMLHttpRequest objektu poslán konkrétní požadavek na server. Server poté provede požadovanou akci a její výsledek, v předem specifikované podobě (např. ve formátu XML), vrátí klientské části aplikace k dalšímu zpracování. Hlavním rysem použití těchto technologií je tedy změna obsahu určité části webové stránky, bez nutnosti jejího kompletního znovunačtení.



Obrázek 3.1: Princip klasické a AJAXové aplikace [12]

3.3.1 Objekt XMLHttpRequest

Objekt XMLHttpRequest (dále jen XHR) představuje základní komunikační rozhraní mezi serverovou a klientskou částí webových aplikací založených nad souborem technologií AJAX.

XHR byl původně implementován ve webovém prohlížeči *Internet Explorer 5* jako komponenta *ActiveX*. Z důvodu implementace pouze v *Internet Exploreru* bylo jeho použití vývojáři webových aplikací poměrně značně omezeno. Tato skutečnost se změnila až po jeho uznání za téměř standard v *Mozille 1.0* a *Safari 1.2*. [1]. Nutno podotknout, že XHR není W3C[8] standard a jeho použití se z tohoto důvodu může v různých prohlížečích mírně lišit.

Jak už bylo řečeno XHR zajišťuje jakési komunikační rozhraní mezi serverovou a klientskou částí webových aplikací. Jeho funkcí je tedy vysílat požadavky na server a zpracovávat přijaté odpovědi. Samotná komunikace je realizována pomocí protokolu HTTP, který pro spojení nabízí metody GET a POST. Hlavním rozdílem těchto metod je, že data odeslaná metodou POST, narozdíl od metody GET, nejsou nijak limitovaná v rámci rozsahu odeslaných dat. Dalším rozdílem je fakt, že data odesílaná metodou GET se odesílají pomocí URL. Pro uživatele tedy nepředstavuje žádný problém tyto data změnit. Z tohoto důvodu by se metodou GET neměli přenášet citlivá data (např. hesla) [1].

[8] *„World Wide Web Consortium je společnou aktivitou jednotlivců i organizací, která si klade za cíl setřídit webové standardy a maximálně jich využít ve prospěch celosvětové komunity“* [31].

Kapitola 4

Frameworky na platformě PHP

Framework (do češtiny překládán jako aplikační, případně vývojový rámec) je softwarová struktura, vytvořená za účelem podpory pro návrh, organizaci, implementaci a v mnoha případech také testování jiných softwarových produktů (aplikací). Pro vývoj aplikací pomocí určitého frameworku bývá používáno jeho API, které definuje způsob práce s frameworkem. Obecně si lze pod označením framework představit jakýsi integrovaný systém složený z mezi sebou komunikujících částí. Tyto části mohou tvořit třídy implementující nejčastěji řešenou problematiku (v daném kontextu použití frameworku), skripty, nebo různé sady knihoven a funkcí. Termín framework je nicméně označení značně obecné a může být používano v různých oblastech s lišícím se významem. V následujícím textu bude termín framework používán výhradně v kontextu softwarového inženýrství.

V dnešní době existuje velké množství frameworků, vytvářených k ulehčení procesu vývoje webových aplikací. Na portálu *codegeekz.com* [8] můžeme najít zhodnocení frameworků dle různých specifik. Je potřeba zmínit, že zmíněné zhodnocení bylo provedeno ke dni 19.srpna 2014 a je samozřejmé, že se v poměrně blízké budoucnosti stane neaktuální. Příčinou jsou neustále vznikající nové frameworky a také fakt, že se u většiny stávájících frameworků postupně aktualizují jejich verze. V této podkapitole budou představeny jedny z nejpoužívanějších frameworků současnosti založených na platformě PHP. Největší pozornost bude věnována možnostem jejich tvorby znovupoužitelných částí kódů. V terminologii představených frameworků půjde o *zásuvné moduly*, *komponenty*, *knihovny*, nebo o tzv. *helpery*.

4.1 Zend Framework

Zend Framework je aplikační rámec pro platformu PHP, šířen jako open-source projekt pod licencí *New BSD*. Tvůrcem tohoto frameworku je firma Zend Technologies, která jeho první verzi s označením *Pre Alpha Version 0.1.1* představila v březnu roku 2006 [5]. Od této verze, která ještě nebyla doporučena pro nasazení do praxe, framework prošel velkým vývojem. Aktuální verze nese označení *2.4.2* a oficiální webové stránky uvádí, že od vypuštění verze *Zend Framework 2* došlo k přes 15 miliónů stažení, což z tohoto aplikačního rámce činí jeden z nejpoužívanějších frameworků dnešní doby. [32]

Zend framework je postaven na návrhovém vzoru MVC a zahrnuje v sobě komponenty pro filtraci dat, validaci dat, práci s formuláři, databází, vyrovnávací pamětí nebo kompo-

nenty pro autentizaci a řízení přístupů. Hlavním rysem frameworku je princip tzv. "use at will" (použít dle potřeby). Díky své modulární architektuře mezi sebou jednotlivé komoponenty obsahují minimální závislosti a mohou se tak používat samostatně, bez použití celého frameworku. Zend framework navíc obsahuje API pro snadné vytvoření nových, případně rozšíření stávajících komponent.

Všechna vlastní rozšíření (tedy i komponenty) se v Zend Frameworku ukládají do adresáře `/library`. Názvy tříd implementující rozšíření jsou podle konvence frameworku složeny z prefixů názvů adresářů, ve kterých jsou v adresáři `/library` uloženy. Název třídy implementující rozšíření komponenty *Zend_Mail* definované v adresáři `library/Own` bude mít název *Own_Mail* a bude uložena v souboru Mail.php. Všechna implementovaná rozšíření se dají používat bez jakékoliv konfigurace pomocí komponenty *Zend_Loader* [5]. Dalším způsobem, jakým lze ve frameworku přistupovat ke znovupoužitelnosti kódu může být pomocí tzv. *Action Helperů*, případně *zásuvných modulů*. Všechny zásuvné moduly a action helpery, které jsou ve frameworku implementovány, jsou podrobně popsány v oficiální dokumentaci[1].

Správu veškerých action helperů má na starost objekt *Action Helper Broker*, kterému je potřeba oznámit pozici nově vytvořeného helperu. Implementace nového helperu může být uložena v adresáři `/library` rozšiřující Zend framework, případně v adresáři aplikace. V prvně zmiňovaném příkladu uložení se objektu *Action Helper Broker* předá pouze prefix [5]. Při uložení v adresáři aplikace je potřeba kromě prefixu předat i úplnou cestu k adresáři, kde se implementace helperu nachází. Samotná třída implementující vlastní action helper musí být potomkem třídy `Zend_Controller_Action_Helper_Abstract`. Podrobný popis, jak action helpery vytvářet a používat, je uveden v oficiální dokumentaci Zend Frameworku.

4.2 CodeIgniter

CodeIgniter[2] je PHP framework založený na návrhovém vzoru MVC. Současná verze tohoto frameworku nese označení *3.0.0* a o jeho hlavní vývoj se stará americká společnost EllisLab. Framework je nicméně šířen jako open-source projekt pod licencí MIT, k jeho rozvoji tudíž velkou mírou přispívá rozsáhlá komunita vývojářů. Mezi hlavní přednosti patří jeho čisté provedení spolu s přehlednou dokumentací, díky níž se tento framework vyznačuje příkrou křivkou učení.

Framework obsahuje celou řadu implementovaných *helperů*[3], řešících např. práci se soubory, emaily, řetězci, formuláři atd. Jednotlivým *helperům* je možné přidávat funkcionalitu, nebo mohou být kompletně přepsány. Ve CodeIgniteru k tomuto účelu stačí definovat prefix, jímž budou upravované helpery identifikovány. Takovéto *helpery* poté CodeIgniter použije přednostně.

Podobným způsobem lze také přizpůsobovat knihovny frameworku. K úpravě funkcionality určité knihovny framework vyžaduje splnění těchto bodů:

1. Třída implementující rozšíření funkcionality, musí být potomkem originální třídy implementující tuto knihovnu. (Tyto třídy jsou označeny prefixem `CI_`)

[1] dostupné na adrese *http://framework.zend.com/*.
[2] Více informací o tomto aplikačním rámci jsou dostupné na adrese *http://www.codeigniter.com/*.
[3] ucelených částí kódu, použitelných napříč celou aplikací

2. Název třídy implementující rozšíření musí být, spolu se souborem, ve kterém je třída definována, označen prefixem `MY_`

Samozřejmostí je rovněž možnost vytvářet *helpery*, nebo *knihovny* zcela nové.

4.3 Nette framework

Aplikační rámec Nette[4] je multiplatformní open-source framework, určený pro tvorbu webových aplikací v PHP 5. Tento framework, jehož původním autorem je český programátor David Grudl, je šířen pod licencí *GNU GPL* a v současné době se o jeho další rozvoj stará organizace Nette Foundation. První oficiální verze s označením *Nette 0.7* vyšla na konci ledna roku 2008. Od této doby prošel framework velkým rozvojem a díky své jednoduchosti a strmé křivce učení si především u českých vývojářů získal velkou popularitu. Podle ankety *The Most Popular Framework of 2015* serveru *sitepoint.com* zveřejněné dne 28. března 2015, se jedná o třetí nejpopulárnější framework vůbec [26].

Mezi největší přednosti Nette patří zejména podpora pro zabezpečení webových aplikací. Pomocí šablonovacího systému *Latte*, s použitím technologie *Context-Aware Escaping*(viz. 4.3), automaticky dochází ke správnému *escapování*[5] vypisovaných dat. Díky této podpoře jsou webové aplikace vytvořené frameworkem Nette odolné vůči bezpečnostním hrozbám jako je *Cross-Site Scripting*[6]. Nette taktéž automaticky řeší ošetřování všech vstupů (na úrovni jednotlivých bajtů) do webové aplikace a tím řeší bezpečnostní hrozby typu *URL attack*. Dalším typem útoků na webové aplikace jsou spojeny se správou *session*[7], kdy útočník pomocí zcizení, nebo podstrčení svého *session ID* (identifikátoru relace), získá přístup do webové aplikace pod konkrétním uživatelem. I tyto typy útoků Nette umí řešit, a to pomocí automatické konfigurace serveru a PHP.

Důležitým milníkem v historii vývoje Nette byl květen 2014, kdy byla vydána verze *Nette 2.2.0*. Tato verze rozdělila do této doby celistvý framework do dvaceti menších projektů, fungujících (relativně) nezávisle na sobě. Tento fakt umožňuje použití jednotlivých částí frameworku jako samostatných komponent. Důležitým faktorem této změny bylo zachování kompatibility s předchozí (celistvou) verzí a tím pádem i zachování způsobu použití frameworku jako celku. V následujících podkapitolách budou popsány jedny z nejdůležitějších částí frameworku.

Šablonovací systém Latte

Šablonovací systém je systém jehož primárním účelem je zjednodušovat generování HTML kódu. Šablona, která definuje zdrojový kód, z něhož šablonovací systém generuje výsled-

[4] Oficiální webová dokumentace je dostupná na adrese *http://nette.org/*. Z této dokumentace byla čerpána většina informací, která jsou uvedená níže.

[5] *„Escapování je převod znaků majících v daném kontextu speciální význam na jiné odpovídající sekvence.“*[23].

[6] *„Cross-site scripting (XSS) útok lze považovat za speciální případ útoků typu Code Injection, při nichž útočník vloží na vybrané místo (vstup aplikace) vlastní zlomyslný obsah (kód), který je následně aplikací přenesen na jiné místo (výstup) a interpretován (spuštěn) v kontextu, který by bez použití této techniky byl útočníkovi jinak nedostupný.“*[21]

[7] Session, neboli relace představuje permanentní spojení mezi serverem a klientem, používané k uchovávání daného stavu, nebo k získání určitých informací (u webového serveru např. o uživatelích, kteří k němu přistupují).

nou HTML podobu představuje (v terminologii architektury MVC) pohled aplikace. Jejím zavedením tedy oddělujeme kód aplikační logiky od reprezentace výstupů. Latte bylo po dlouhou dobu[8] integrovaná součást frameworku Nette. V roce 2014 se stalo, v rámci rozsáhlého rozložení framewroku na menší části, samostatnou komponentou. Díky tomu je jeho použití možné i na projektech a webových aplikací nevyvíjených nad frameworkem Nette.

Hlavní předností Latte je tzv. *Context-Aware Escaping*, technologie automatického escapování, která při vypisování proměnných detekuje jejich kontext a na jeho základě zvolí správnou escapovací strategii.

Syntaxe Latte vychází z klasické syntaxe složených závorek, používané u většiny šablonovacích systémů. Kromě toho umožňuje zápis pomocí tzv. *n:maker*, které zpřehledňují kód při použití řídících konstrukcí jako jsou cykly nebo iterace. Další užitečnou pomůckou v Latte jsou *filtry* (někdy označované jako *helpery* či *modifikátory*). Filtry jsou funkce zapisující se za "svislítkem", sloužící pro upravení nebo přeformátování dat do výsledné podoby. Následující příklad znázorňuje použití *n:makra* s filtry pro převod řetězce na velká písmena (*upper*) a odstranění diakritiky (*toAscii*).

```
<span n:foreach="$items as $item">{$variable|upper|toAscii}</span>
```

Další kladnou stránkou šablonovacího systému Latte je jeho rychlost. Jednotlivé šablony jsou překládány do nativního kódu PHP a ukládány do vyrovnávací paměti na disk. Na vysoké úrovni je také koncepce podpory dědičnosti šablon, nebo spolupráce s ladícím nástrojem *Tracy*, jenž bude popsán v následující podkapitole.

Tracy

Knihovna *Tracy*, do verze *Nette 2.2.0* označována jako *Nette\Diagnostics\Debugger*, je užitečný nástroj určený pro ladění webových aplikací. Vyznačuje se vizuálně přehledným výpisem výjimek, chyb nebo ladících informací. Obsahuje *Debugger Bar*, zobrazující užitečné ladící informace a měřič času pro změření trvání náročnějších operací. Tracy se dá navíc snadno napojit na *FireLogger* prohlížeče *Firefox* a s jeho pomocí zobrazovat ladící informace týkající se např. AJAXových voláních.

Tester

Nette Tester je nástroj pro tvorbu automatických testů vytvořený na platformě PHP. Jeho autorem je David Grudl a mezi jeho největší přednosti patří jednoduchost používání, možnost krokování v IDE[9] (integrovaná podpora v *NetBeans 8.0*) nebo paralelní spouštění testů. Spouštění testů je prováděno z příkazového řádku a to tím způsobem, že se pro každý test spustí nový PHP proces (jednotlivé testy probíhají nezávisle na sobě). Pro vyhodnocování testů slouží tzv. *Aserce*, což jsou metody třídy `Tester\Assert`, používané pro srovnávání očekávané hodnoty s hodnotou, jenž vrátí daný test na základě určitého vstupu. Tyto *aserce* lze pomocí třídy `Tester\TestCase` strukturovat do různých sad testů. *Nette*

[8] od roku 2008, kdy bylo oficiálně představeno *Nette 0.7*

[9] *Integrated Development Environment* (IDE), neboli vývojové prostředí je software usnadňující vývoj aplikací vytvářených v určitém programovacím jazyce. Běžně obsahuje editor zdrojového kódu, *kompilátor* případně *interpret* pro překlad algoritmů do nižšího programovacího jazyka (strojového kódu) a převážně také *debugger* s krokovacím nástrojem pro odhalování chyb.

Tester umožňuje mimo jiné zápis pomocí tzv. *anotací*[10]. *Anotace* se zapisují na začátek souboru, jejich název vždy začíná @ a slouží pro rozšíření funkcionality jednotlivých testů (sad testů), nebo k doplnění určitých informací. V následujícím příkladu zápisu *anotace* pro test s názvem *Základní test*, je použit zápis pro definici očekávaného návratového kódu (`@exitCode 10`), opakované spuštění testu (`@multiple 10`) a podmínění spuštění testu pouze v případě určité verze PHP (`@phpVersion < 5.4.0`).

```
/**
* TEST: Zakladni test.
*
* @exitCode 10
* @multiple 10
* @phpVersion < 5.4.0
*/
```

Jako další užitečnou podporu pro vytváření testů pomocí nástroje *Nette Tester* lze uvést třídu `Tester\DomQuery` (implementovanou za účelem usnadnění testování HTML, případně XML souboru), nebo třídu `Tester\FileMock`, která emuluje soubory v paměti.

4.3.1 Podpora technologií AJAX

Problematika technologií s označením AJAX a jejich přínosy pro tvorbu interaktivních webových aplikací byla popsána v kapitole 3.3. V této kapitole bude znázorněn postup, jak lze v Nette vytvářet AJAXové požadavky. Způsobů může být více s mírně se lišícím postupem, zde je uveden postup, kterým byly implementovány AJAXové komponenty, jenž jsou součástí této bakalářské práce.

Klientská strana

Framework Nette na straně klienta neposkytuje žádnou podporu pro tvorbu AJAXových požadavků. K jejich implementaci zároveň nevyžaduje žádnou konkrétní javascriptovou knihovnu nebo framework. Volba je tedy zcela ponechávána na vývojáři. Jedna z možností je použití javascriptové knihovny JQuery s Nette framework pluginem *jquery.nette.js*, jejímž autorem je Jan Marek. Obsluha klientské strany AJAXové webové aplikace se skládá z těchto kroků:

1. Definice HTML elementů, které budou iniciovat AJAXový požadavek

   Nejelegantněji lze splnění tohoto kroku docílit pomocí tzv. *unobtrusive* [11] *JavaScript*, kdy je na jednotlivé elementy, určené k iniciování AJAXového požadavku přiřazena určitá třída (podle konvence např. třída *ajax*). Funkcí této třídy je identifikace elementu, na kterém budou "zavěšeny" javascriptové funkce implementující komunikaci se serverem.

   ```
   <a href="link getOtherPage!, 2" class="ajax" >... </a>
   ```

[10] Anotace představují speciální druh komentářů, zapsaných do *phpDoc* bloků. Úkolem těchto komentářů je rozšiřovat schopnosti PHP o další funkcionalitu.

[11] Technika umožňující definovat události iniciující konkrétní javascriptové funkce mimo strukturu HTML.

Ajaxový požadavek je v tomto případě iniciován přes tzv. *signál*, kterým je volána metoda `getOtherPage()`.

2. Odeslání reakce na server

   K odeslání požadavku na server může být použita funkce `get([settings])` knihovny JQuery, která slouží k načtení dat ze serveru použitím HTTP GET požadavku. V předešlém kroku byla iniciace AJAXové požadavku "navázána" na element $<a>$ v HTML představující odkaz. Protože jsou AJAXové požadavky vykonávány bez znovunačtení stránky, je potřeba všechny tyto odkazy zneplatnit. K tomuto účelu slouží JQuery funkce `event.preventDefault()`. Kompletní odeslání reakce na server může vypadat následovně:

   ```
   $(document).on("click","a.ajax", function (event) {
       event.preventDefault();
       $.get(this.href);
     }
   });
   ```

   K zabránění opětovného odeslání AJAXového požadavku může posloužit technika skrytí HTML elementu, iniciujícího daný požadavek:

   ```
   $("a.ajax").css("visibility","hidden");
   ```

   Po dokončení AJAXového požadavku, který je možný detekovat pomocí JQuery funkce `ajaxStop()`, bude element opět zobrazen.

3. Zpracování odpovědi vrácené serverem

   Ke zpracování odpovědi vrácené serverem slouží již zmíněný Nette framework plugin *jquery.nette.js*. Hlavní funkcí tohoto pluginu je aktualizace obsahu *snippetů* určených k překreslení.

Serverová strana

Na serverové straně webové aplikace Nette poskytuje úložiště *payload*, které je součásti každého presenteru[12]. Do tohoto úložiště lze zapisovat jakákoliv data a presenter je poté pošle na výstup serializované. Typickým formátem, používaným pro serializaci, je formát JSON. Úložiště *payload* používají i *snippety*, které do tohoto úložiště ukládají svůj obsah.

1. Označení HTML fragmentů stránky k překreslení

   Asi největší předností vytváření AJAXových webových aplikací, pomocí frameworku Nette (ve smyslu urychlení vývoje), je použití tzv. *snippetů*. Pojmem *snippet* je v Nette označován fragment HTML, který je určen k přenosu dat mezi klientskou a serverovou částí webové aplikace. K vymezení obsahu *snippetů* slouží párové makro {`snippet`}. Šablonovací sytém *Latte* makro {*snippet nazev*} ... {*/snippet*} ve výsledné HTML podobě nahradí konstrukcí $<$*div id="snippet–nazev"* $>$... $<$*/div*$>$. Změnu elementu

[12] Za *presenter* je v terminologii Nette frameworku označován soubor, představující implementaci aplikační logiky webové aplikace.

div za jakýkoliv jiný element, je možné provést uvedením jeho názvu za názvem *snippetu* (např. {*snippet nazev span*}). *Snippetů* může být na jedné stránce více. K detekci toho, jaký *snippet* má být přenesen a jaký má být naopak ponechán, slouží mechanismum, v terminologii Nette označován jako *zneplatnění*. K zneplatnění slouží metoda `invalidateControl()`, která jako volitelný parametr přijímá název *snippetu*. Obsah zneplatněného *snippetu* je po zpracování požadavku aktualizován (platné *snippety* se naopak nepřenášejí a zůstávají nezměněny). Pokud je metoda `invalidateControl()` volána bez parametru, jsou všechny *snippety* daného presenteru považovány za neplatné a přenesou se tedy všechny.

2. Detekce AJAXového požadavku s případným přesměrováním

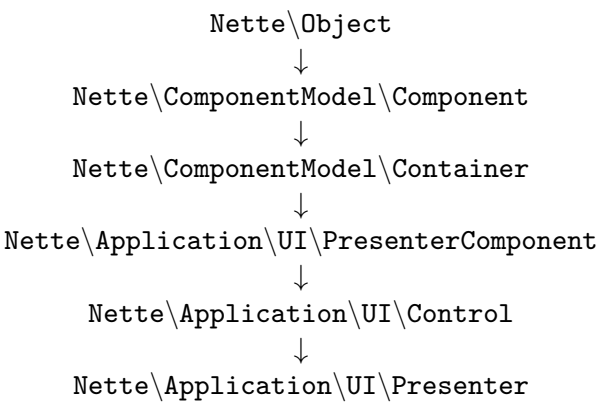
   K rozeznání AJAXového požadavku od klasického, slouží metoda `isAjax()`. Pomocí této metody můžeme určit, zda se má po provedení daného požadavku stránka přesměrovat, nebo nikoliv.

4.3.2 Komponenty a jejich tvorba

Komponenta představuje znovupoužitelnou ucelenou část kódu, vytvořenou pro vizuální případně čistě funkcionální účel. Obecně řečeno se jedná o pojmenovaný objekt, jehož základním předpokladem je možnost být součástí jiného pojmenovaného objektu. Komponenty tvoří základní kámen znovupoužitelnosti kódu a jejich používání ve webových aplikacích značně zpřehledňuje zdrojový kód a urychluje celkový vývoj. V této podkapitole bude nastíněna implementace a následné použití komponent ve frameworku Nette.

Základní hierarchie tříd

Nette pro tvorbu komponent využívá API jehož kompletní popis můžeme nalézt v originální dokumentaci. Základní třídou, pomocí které komponenty v Nette vytváříme, je třída `Nette\Application\UI\Control`. Tato třída je v hierarchii dědičnosti potomkem třídy `Nette\Application\UI\PresenterComponent`, tvořící základní třídu všech komponent daného presenteru. Komponenty v presenteru představují persistentní objekty, jež si presenter uchovává v průběhu svého životního cyklu. Potomkem této třídy je třída `Nette\ComponentModel\Container`, obsahující metody pro manipulaci s komponentami (např. metody pro získávání, přidávání nebo rušení daných komponent). Všechny třídy poté zastřešuje třída `Nette\ComponentModel\Component`, jejíž hlavní funkcí je vytvoření příbuznosti, neboli vytvoření hierarchie daných komponent.

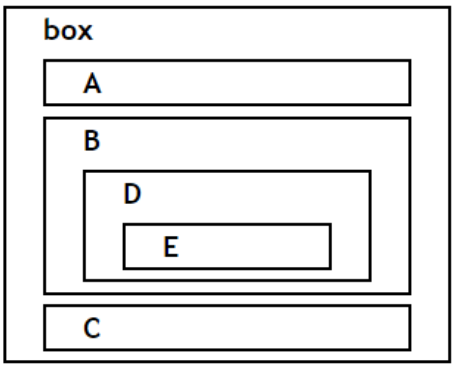


Vytvoření komponenty

Jak už bylo zmíněno, o vytvoření komponenty se v Nette stará třída `Nette\Application\UI\Control`. Z tohoto důvodu bude třída obsahující definici funkcionality dané komponenty potomkem právě této třídy. Třída `Nette\Application\UI\Control` povoluje definici pohledu (HTML výstupu) v externím souboru .latte, to umožňuje v metodě `render()`, sloužící k vykreslení dané komponenty, uvést soubor, ze kterého se bude načítat šablona komponenty. Následující kód ukazuje vytvoření komponenty definované ve třídě `ComponentControl`, pomocí tzv. *továrničky*, tedy funkce v Nette implementované výhradně za účelem vytváření komponent. Na obrázku 4.1, reprezentující HTML výstup výsledného seskupení komponent, je možné vidět, jak lze tyto jednotlivé komponenty zapouzdřovat do sebe a vytvářet tak komplexní webové prvky.

```
protected function createComponentBox(){
  //vytvoření instance třídy implementující danou komponentu
  $control = new ComponentControl();
  //do komponenty můžeme přidávat i další komponenty
  $control->addComponent(new ComponentControl, 'A');
  $control->addComponent(new ComponentControl, 'B');
  $control->addComponent(new ComponentControl, 'C');
  //příklad zkráceného zápisu pro získání komponenty
  $b = $control['B'];
  //a následného přidání další komponenty
  $b->addComponent(new ComponentControl, 'D');
  $d = $control['B-D'];
  $d->addComponent(new ComponentControl, 'E');

  return $control;
}
```



Obrázek 4.1: Příklad zanořování komponent.

Samozřejmostí je také možnost používání odkazů *link*, které v Nette fungují jako odkazy na konkrétní funkce presenteru (obvykle na metody `handle`*Název_signálu*). Použití těchto odkazů je možné v definici funkcionality komponenty nebo v šabloně.

Použití v presenteru	Požití v šaboně
`$this->link('click!', $x, $y)`	`<a n:href="click! $x,$y">`

Kde `$x` a `$y` jsou parametry předávané pomocí URL. Pro složitější komponenty je možné použití tzv. *flash zpráv*, nebo *snippetů* (viz. 4.3.1).

Kapitola 5

XTemp preprocesor

XTemp preprocesor byl vytvořen jako doplněk k frameworku Nette ke snadné implementaci komponent. Umožňuje vytvoření nových komponent a jejich snadné použití v šabloně (XHTML souboru) definující pohled webové aplikace. Je napsán v programovacím jazyce PHP a jeho autorem je Ing. Radek Burget, Ph.D.. Funkcí XTemp preprocesoru je transformace XHTML kódu, definujícího tzv. pohled aplikace. Výsledek transformace představuje řetězec, jenž je předán šablonovacímu systému Latte k vykreslení do výsledné HTML podoby. V následujících kapitolách bude popsán základ funkcionality nutný k pochopení toho, jak XTemp preprocesor funguje.

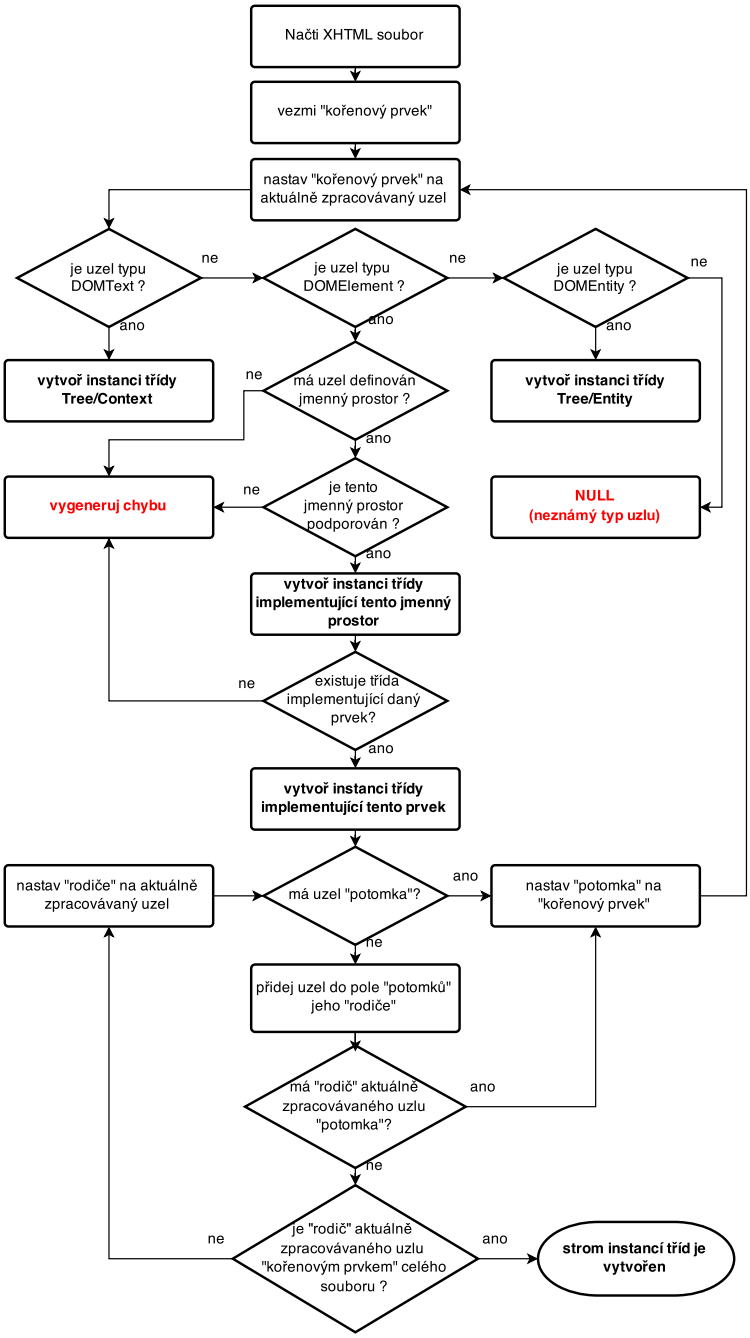
5.1 XTemp a jmenné prostory

Pro zabránění vzniku konfliktů mezi různými formáty, ke kterým by mohlo vést použití komponent implementovaných nad XTemp preprocesorem, je každé komponentě přiřazen jmenný prostor v němž je implementována. Úkolem metody `buildNamespaceTable()` je vytvořit seznam všech těchto použitých jmenných prostorů. Tato metoda je součástí třídy `Filter` implementované v souboru `Filter.php` a jejím výstupem je asociativní pole, obsahující tzv. URI (Uniform resource identifier)[1] všech definovaných jmenných prostorů doposud implementovaných komponent. Ke každému URI je přiřazen název třídy, která daný jmenný prostor definuje. Toto spojení je v dalších částech procesu transformace XHTML souboru použito pro efektivnější vyhledávání implementace vykreslení konkrétního elementu.

5.2 Zpracování XHTML souboru

Funkcionalitu zpracování XHTML souboru má na starost třída `Filter` implementována v souboru `Filter.php`. Hlavní činností této třídy je, pomocí metod, jejichž definice obsahuje, vytvořit tzv. strom instancí tříd. Strom instancí tříd je, jak už název napovídá, datová struktura ve formě stromu, obsahující instance tříd jednotlivých komponent. Jakým způsobem se tento strom vytvoří popisuje vývojový diagram 5.1. Seznam veškerých metod implementujících funkcionalitu zpracování XHTML souboru je spolu s jejich popisem uveden v příloze D tohoto dokumentu.

[1] *„Uniform Resource Identifier (URI) je kompaktní řetězec znaků pro identifikaci abstraktního nebo fyzického zdroje.“* [4]



Obrázek 5.1: Zpracování XHTML souboru.

5.3 Princip vykreslování komponent

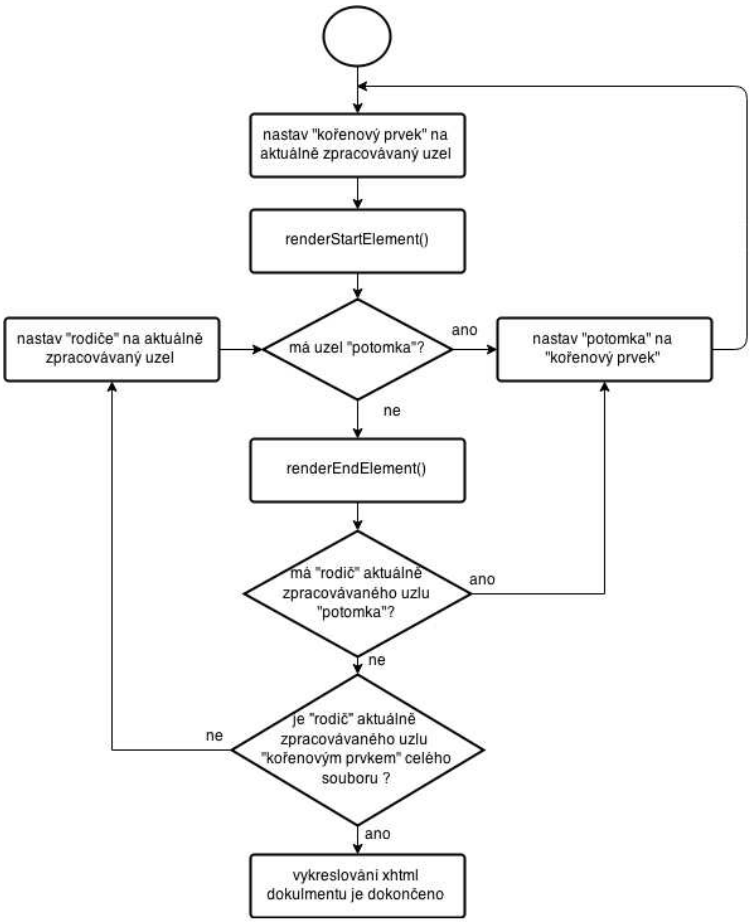
Vykreslování jednotlivých komponent je založeno na vytvořeném stromu instancí tříd. Před samotným vykreslováním komponent metodou `render()` jsou volány metody `beforeRender()`, které obvykle obsahují funkcionalitu jednotlivých komponent, kterou je nutno vyvolat před

finálním vykreslením. Posloupnost volání těchto metod je znázorněna následujícím zdrojovým kódem.

```
public function prepareRendering($root)
{
    foreach ($root->getChildren() as $child)
      $this->prepareRendering($child);

    $root->beforeRender();
}
```

Ke tvorbě nových komponent XTemp preprocesor obsahuje API, přizpůsobené k jejich snadnější implementaci vykreslení. Definici metod vytvořených k tomuto účelu obsahuje abstraktní třída `Element`, která je v hierarchii dědičnosti tříd potomkem třídy `Component`. Jednoduchý příklad vykreslení celého XHTML dokumentu popisuje následující vývojový diagram 5.2.



Obrázek 5.2: Princip vykreslování komponent.

5.4 Napojení na Nette framework

Jako tzv. spojovací prvek mezi Nette frameworkem a XTemp preprocesorem slouží třída `XTempPresenter`, která v terminologii Nette představuje BasePresenter (základní presenter, od kterého dědí všechny ostatní presentery) celého XTemp preprocesoru. Aby mohl XTemp preprocesor korektně vykreslovat jednotlivé elementy použité v tzv. šabloně, je potřeba, aby byl BasePresenter dané webové aplikace potomkem právě této třídy. Dalším vstupním prvkem Xtemp preprocesoru můžeme chápat třídu `XhtmlPresenter`. Protože Nette framework defaultně pracuje s šablonami s příponou *.latte*, narozdíl od XTemp preprocesoru, který jako datový vstup očekává soubory typu XHTML, je potřeba toto defaultní načítání šablon změnit. Právě k tomuto účelu slouží třída `XhtmlPresenter`, která přepisuje metodu `formatTempateFiles()`, jenž se v Nette stará o dohledání šablon.

5.5 Implementované rozšíření XTemp preprocesoru

V rámci této bakalářské práce byl XTemp preprocesor rozšířen o metody umožňující AJAXovou komunikaci komponent *XTempFrame* a *XTempTable*. Rozšíření bylo provedeno z důvodu použití tzv. *signálů*, jimiž je iniciován AJAXový požadavek. *Signály* se v aplikačním rámci Nette používají pro komunikaci šablon s určitými presentery aplikace (více viz 4.3.1). Z tohoto důvodu bylo potřeba metody, které jsou pomocí těchto signálů volány implementovat v `XTempPresenteru`. Konkrétně jde o metody `createComponentXTempForm()`[2], `XTempFormSucceeded()`[2], `setXtempCode()`[2] a `handleOtherPage()`[3]. Tyto metody nijak nepřispívají k celkové funkcionalitě XTemp preprocesoru a jejich použití slouží pouze pro výše zmíněné komponenty.

[2]Funkcionalita metody je nastíněna v příloze C.5 popisující API komponenty *XTempFrame*.
[3]Funkcionalita metody je nastíněna v příloze C.1 popisující API komponenty *XTempTable*.

Kapitola 6

Návrh a implementace sady komponent

Všechny komponenty byly navrženy s cílem snadného a co nejflexibilnějšího použití. Zároveň byl kladen důraz na možnost úpravy jak funkcionálního, tak vizuálního charakteru jednotlivých komponent do požadované podoby. Funkcionální úpravy lze provádět pomocí implementovaných atributů. Z důvodu přehledné a logické strukturalizace grafických částí komponent, lze jejich vizuální charakter snadno optimalizovat pomocí CSS.

Implementace komponent byla provedena pomocí aplikačního rámce *Nette* verze *2.2.1* s použitím preprocesoru XTemp. Všechny komponenty jsou implementovány responzivním[1] přístupem. Díky faktu, že se vzhled těchto komponent přizpůsobuje použitému zobrazovacímu zařízení, je jejich užití možné jak pro aplikace vytvářené pro zobrazovací zařízení větších rozměrů, tak pro aplikace vytvářené pro mobilní zařízení. V následujících kapitolách bude popsáno použití navržených komponent, jejich implementace a způsob jejich funkcionálního, či vizuálního přizpůsobení do požadované podoby.

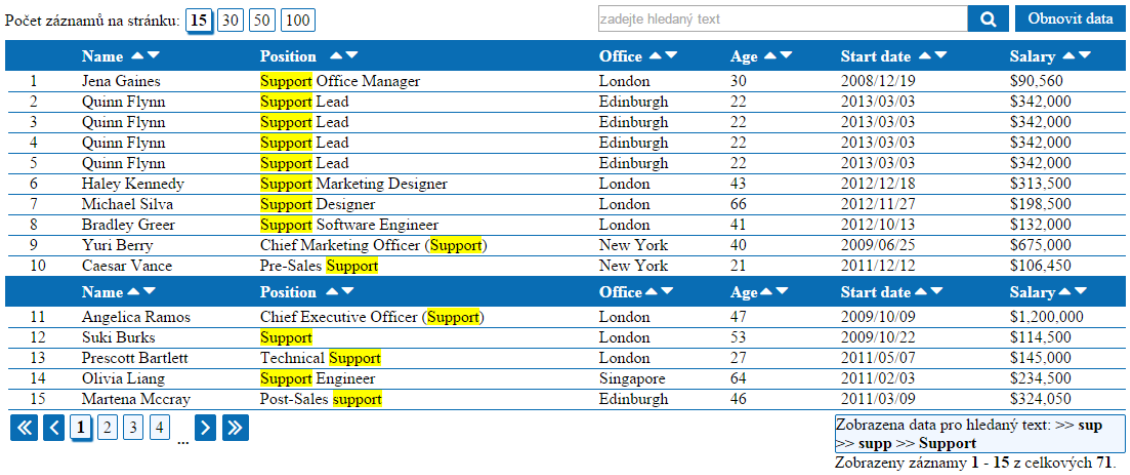
Všechny atributy, používané k funkcionální úpravě navržených komponent, jsou nepovinné. Při chybné definici, atributu např. zvolením nekorektního datového typu, nebo špatného formátu, je automaticky použita defaultní hodnota. Na základě faktu, že jsou všechna nastavení v rámci změny hodnoty atributu vizuálního charakteru, není potřeba na použití defaultní hodnoty atributu uživatele nijak upozorňovat.

6.1 Komponenta XTempTable

Komponenta byla navržena pro snadnou tvorbu datových tabulek založených na AJAXovém načítání dat. Při defaultním nastavení podporuje víceúrovňové vyhledávání, stránkování a řazení záznamů. Uživatel si při vytváření tabulky touto komponentou může definovat a pomocí atributů následně upravovat další implementované funkce.

Mezi tyto funkce patří například opakované vykreslování hlavičky tabulky, skrývání dodatečných informací jednotlivých záznamů tabulky, nebo možnost volit typ řazení v rámci jednotlivých sloupců. Všechny tyto funkce, jejich atributy a způsob implementace jsou konkrétně popsány v následujících podkapitolách.

[1] Jako responzivní web se rozumí takový web, jehož stylování je optimalizováno pro zobrazovací zařízení s různým rozlišením.



Obrázek 6.1: Komponenta XtempTable.

6.1.1 Přehled tagů

Pro snadné použití se tvorba tabulky pomocí komponenty XTempTable opírá o strukturu klasické HTML tabulky. Jediným rozdílem je nahrazení tagu <table>, který uzavírá celou strukturu tabulky tagem <XTempTable>. Kromě standardních HTML tagů, určených pro tabulku, je v komponentě možné použít speciální tag <slide>. Párový tag <slide> skrývá dodatečnou informaci v rámci řádku tabulky. Řádek tabulky, v němž je tento párový tag obsažen, je po vygenerování do výsledné podoby vizuálně odlišen od ostatních řádků bez tohoto tagu. Tento tag je nepovinný a jeho použití je možné kdekoliv mezi párovými tagy <tr> definující řádek tabulky.

Tabulka 6.1: Atributy pro <th> komponenty XTempTable

Atribut	**Hodnota**	**Popis**
`sortingType`	*sort_regular* |*sort_numeric* |*sort_string* |*sort_locale_string* |*none*	Definuje styl řazení položek v daném sloupci. **sort_regular** - Standardní řazení položek (Standard ASCII). **sort_numeric** - Řazení položek podle numerické hodnoty. **sort_string** - Řazení položek podle řetězcové hodnoty. **sort_locale_string** - Řazení položek podle národních standardů. **none** - Odstraní možnost řazení pro daný sloupec.

Tabulka 6.2: Atributy pro <`xtempTable`>

Atribut	**Hodnota**	**Popis**
`itemsPerPage`	*number* |*number2* |*number3* |...	Definuje množinu hodnot určených pro výběr počtu vypsaných řádků tabulky na jednu stránku. Jako separátor je použit středník (;). Defaultní hodnota při absenci atributu je **15;30;50;100**. Tato hodnota se nastaví i tehdy, pokud některá ze zadaných hodnot nebude numerická.
`showLineNumbers`	*yes* |*no*	Zobrazí čísla řádků tabulky. Defaultní hodnota při absenci atributu je **yes**.
`showItemsPerPage`	*yes* |*no*	Zobrazí množinu hodnot určených pro výběr počtu vypsaných řádků tabulky na jednu stránku. Defaultní hodnota při absenci atributu je **yes**.
`showPaginator`	*yes* |*no*	Zobrazí "paginator" pro přechod na další stránky. Defaultní hodnota při absenci atributu je **yes**.
`showInfo`	*yes* |*no*	Zobrazí informaci o počtu aktuálně zobrazených záznamů z celkového počtu záznamů. Defaultní hodnota při absenci atributu je **yes**.
`showSearchItems`	*yes* |*no*	Zobrazí input text pro hledání záznamů v tabulce. Defaultní hodnota při absenci atributu je **yes**.
`itemsInPaginator`	*number*	Definuje maximální počet zobrazených aktivních tlačítek, používaných k procházení stránek v *paginatoru*, v posloupnosti od tlačítka reprezentujícího aktuálně načtenou stránku. Defaultní hodnota při absenci atributu je (3). V jednom okamžiku tedy bude načteno vždy maximálně 7 aktivních tlačítek (3 tlačítka v obou směrech posloupnosti od aktuálně načtené stránky).
`headRepeat`	*number*	Definuje opakované vykreslování hlavičky tabulky. Hodnota tohoto atributu představuje počet záznamů po kterých bude hlavička opětovně vykreslena. Defaultní hodnota při absenci atributu je **0**.

6.1.2 Použití komponenty

Při návrhu komponenty byl kladen důraz na přehlednost a intuitivní použití. Při defaultním nastavení se okolo tabulky zobrazí celkem čtyři aktivní prvky (jak je vidět z obrázku 6.1). V levé horním rohu je to jednoduchý selektor s možnostmi pro výběr počtu zobrazených záznamů na jedné stránce. V pravém horním rohu jsou umístěny aktivní prvky pro víceúrovňové vyhledávání. Pod spodní hranou tabulky je na levé straně umístěn *paginator* pro ovládání stránkování. Informace o filtrováních a počtech zobrazených záznamů je umístěna v pravém dolním rohu. Podobu a umístění jednotlivých prvků je možné přizpůsobit jednoduchou úpravou pomocí CSS.

Víceúrovňové vyhledávání

Jako vstupní bod pro vyhledávání slouží *"input box"* pro zadávání hledaného řetězce. Po potvrzení potvrzovacím tlačítkem jsou uživateli AJAXově načtena data. Pokud zadanému řetězci neodpovídá žádný hledaný text, je uživateli o této skutečnosti zobrazena stručná informace. Hledaný řetězec, pomocí kterého byla data filtrována, je v nově načtených datech vizuálně vyznačen. S filtrovanými daty je možné pracovat stejně jako s nefiltrovanými. Tedy procházet jednotlivé stránky, volit počet zobrazených záznamů nebo řadit v rámci jednotlivých sloupců. Uživatel má také možnost nad již zfiltrovanými daty provést další filtraci podle zadaného řetězce. Tato filtrace se může provádět opakovaně. Sekvence řetězců, které byly použity pro filtraci, spolu s informací o počtu nalezených záznamů, je uživateli oznámena v přehledném informační boxu. Po stisknutí tlačítka "Obnovit data" jsou načtena původní data definovaná v XHTML souboru.

6.1.3 Popis implementace

Komponenta je implementovaná v souborech `XtempTableElement.php` (implementace vykreslení při prvním načtení stránky) a `XtempTableAjaxRender.php` (implementace vykreslení při AJAXových požadavcích), jejichž API je popsané v příloze C.1 tohoto dokumentu. XTemp preprocesor nejdříve vytvoří instanci třídy `XtempTableElement`, ve které volá popořadě metody `beforeRender()` a `Render()`, tak jak bylo popsáno v kapitole 5.3. Metoda `Render()`, pomocí metod implementovaných ve třídě `XtempTableElement`, vytvoří řetězec, který předá šablonovacímu systému Latte pro první vykreslení datové tabulky. Všechna následující AJAXová vykreslování, po prvním načtení stránky, mají na starost metody třídy `XtempTableAjaxRender`.

AJAX

AJAXová komunikace je u komponenty implementovaná klasickým "Nette" způsobem. O tom jakou Nette poskytuje podporu pro AJAXovou komunikaci, je podrobně popsáno v kapitole 4.3.1. Protože Nette samotné neobsahuje implementaci AJAXových požadavků na straně klienta, je použita knihovna *jquery.nette.js*, jejímž autorem je Jan Marek a je šířena pod licencí MIT. Části HTML, které je potřeba překreslovat jsou obaleny párovým makrem snippet. Toto makro zajistí, že se nepřekreslí celá stránka, ale pouze html fragment, jenž makro obaluje. Ajaxový požadavek je iniciován na straně klienta, kde je přes tzv. signál poslán metodě `handleOtherPage()`. Metoda `handleOtherPage()` z požadavku extrahuje potřebná data a předá je metodě `GetOtherPage()`. Tato metoda podle přijatých parametrů provede požadovanou akci a výsledná data předá opět přes metodu `handleOtherPage()` šabloně k vykreslení.

Víceúrovňové vyhledávání

Víceúrovňové vyhledávání je implementováno v metodě `getFilteredSearchText()` třídy `XtempTableAjaxRender`. V této metodě se pomocí klasického *foreach* cyklu prochází textové hodnoty jednotlivých sloupců. Pokud se v nějakém sloupci hledaný řetězec nachází, obalí se třídou pro zvýraznění hledaného řetězce a záznam, jemuž náleží, se uloží do nově vytvořeného pole s výslednými záznamy. Podle identifikátoru, který je přiřazen každému záznamu, se vyhledá *slide* záznam a přidá se k vyfiltrovanému záznamu, kterému náleží. Poté se nastaví příznak *filtered* označující, že data byla filtrována. Tento příznak při dalším požadavku zajistí, aby se pracovalo s filtrovanými daty namísto s původními. Nakonec se aktuálně hledaný text přidá do řetězce k vykreslení sekvence hledaných textových hodnot v informačním boxu. S každým dalším požadavkem se kontroluje identifikátor *filtered*, jehož hodnotu z hodnoty *true* na *false* mění uživatel stisknutím tlačítka *Obnovit data*. Pokud identifikátor *filtered* nabývá hodnoty *false,* načítají se původní data, namísto filtrovaných a ukončuje se tak sekvence víceúrovňového vyhledávání.

Řazení záznamů

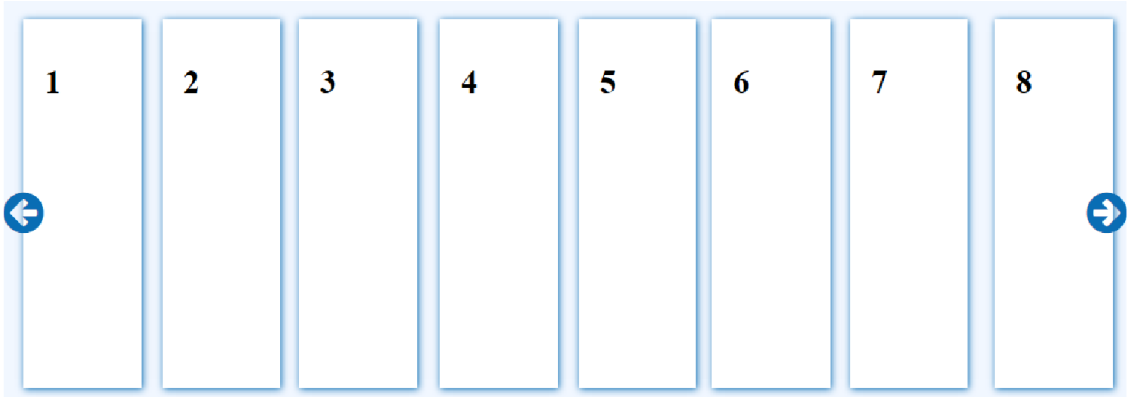
Řazení záznamů je založeno na standardní PHP funkci `array_multisort`, která umožňuje řadit vícenásobné, nebo vícerozměrné pole dle zadané hodnoty a zadaného typu řazení. Metoda `getOtherPage()` podle proměnné *column*, kterou dostane na vstupu, zjistí, podle jakého sloupce mají být data seřazena a vytvoří pole pouze s položkami tohoto sloupce. Metoda `getSortArray()` poté pomocí tohoto vytvořeného pole spolu s proměnnou *sortType* (určující typ řazení) a proměnnou *order* (určující směr řazení) seřadí záznamy a vrátí seřazená data připravená k vykreslení. Po seřazení jednotlivých záznamů je volána metoda `setSortedRows()`, jenž se postará o správné přiřazení *slide* záznamů. Nakonec metoda `getOtherPage()` původní data přepíše nově seřazenými daty. Díky tomu je nad těmito daty možné provádět stránkování a další funkce komponenty XtempTable, jako by se jednalo o data původní.

Stránkování

Dynamické překreslování aktivních prvků *paginatoru* má na starost metoda `getPaginator()` třídy `XtempTableElement`. V metodě se nejprve na základě celkového počtu záznamů v tabulce, k aktuálně zvolenému počtu zobrazených záznamu na stránce, vypočítá celkový počet stránek. Na základě atributu *ItemsInPaginator* se zvolí maximální počet v jednom okamžiku zobrazených aktivních prvků a spolu s posuvníky se přidají do výsledného řetězce k vykreslení. Pokud se jedná o první načtení stránky, parametry a proměnné potřebné k předání metodě `getOtherPage()` se generují ve třídě `XTempTableElement`. Při AJAXovém požadavku jsou proměnné definovány v metodě `handleOtherPage()` XTemp presenteru a předány přímo do Latte šablony.

6.2 Komponenta Slider

Komponenta Slider umožňuje snadně implementovatelnou a vizuálně efektivní prezentaci textů, obrázků, nebo jakéhokoliv HTML obsahu ve formě tzv. "slideshow". Při návrhu komponenty byl kladen důraz na intuitivní a snadnou konfiguraci z pohledu uživatele. Veškerá funkcionalita lze snadno upravovat pomocí atributů, které jsou popsány v tabulce 6.3. Použitím atributů v kombinaci spolu s CSS lze výslednou "slideshow" velice snadno upravit do požadované podoby.



Obrázek 6.2: Komponenta Slider.

6.2.1 Popis implementace

Komponenta Slider je implementována v souborech `SliderBox.php` a `SliderElement.php`. Z důvodu, že je implementace této komponenty z převážné většiny tvořena kódem zajišťujícím funkcionalitu zpracování jednotlivých atributů, bude z důvodu přehlednosti popsána právě na těchto atributech. V následující výčtu bude tedy ke každému atributu popsán způsob jakým bylo dosaženo jeho funkcionality. Kompletní API komponenty Slider je popsáno v příloze C.2 tohoto dokumentu.

Atribut	**Popis implementace**
`slideInterval`	K implementaci funkcionality atributu slideInterval předcházela implementace rotace boxů, ovládaná šipkami. Na každou ze dvou šipek byla vytvořena funkce `leftRotate()`, případně `rightRotate()` (podle směru rotace boxu). Automatická rotace boxu je implementována pomocí JQuery funkce `setInerval($funkce,$interval)`. Zdrojový kód demonstrující implementaci rotaci boxů na pravou stranu je vyobrazen v příloze B této bakalářské práce.
`slideNumber`	K implementaci rotace o určitý počet elementů jsou použity JQuery funkce `prev()` (pro rotaci elementů na levou stranu) a `next()` (pro rotaci elementů na pravou stranu). Tyto funkce jsou určeny pro selekci HTML elementů, konkrétně pro posun na sousední element. Použití těchto funkcí můžeme vidět na demonstrovaném kódu B.

`box` Stejně jako všechny ostatní navržené komponenty má i komponenta Slider responzivní charakter. Šířka boxu ohraničující rotující elementy komponenty Slider je nastavena na hodnotu 100%, jeho velikost se tedy bude odvíjet od velikosti rodiče (elementu v němž je komponenta Slider umístěna). Ke zobrazení určitého počtu v jednom okamžiku zobrazených boxů, bylo potřeba vypočítat šířku těchto elementů. Šířka jednotlivých elementů byla vypočtena podle vzorce:

$$sirka\_elementu \; = \; 100 \; / \; pocet\_elementu \; - \; okraj\_elementu$$

a nastavena pomocí JQuery funkce `css()`, která umožňuje dynamicky optimalizovat CSS stylopis.

`speed` Jak je možné vidět na přiloženém kódu B, extrahovaná hodnota atributu *speed* se použije ve JQuery funkci `animate()`.

`automaticWay` Automatická rotace elementů je implementována pomocí JQuery funkce `setInterval(leftRotate | rightRotate, interval)`. Tato funkce slouží k opětovnému spouštění funkce, určené prvním parametrem (výběr funkce se provede na základě atributu *automaticWay*). Jako parametr *interval*, určující časový interval spouštění funkce, je této funkci předána extrahovaná hodnota atributu *slideInterval*.

Tabulka 6.3: Atributy pro <`sliderElement`>

Atribut	**Hodnota**	**Popis**
`box`	*number*	Určuje počet v jednom okamžiku zobrazených boxů v komponentě Slider. Defaultní hodnota při absenci atributu je **3**.
`slideInterval`	*number* | *none*	Definuje čas v milisekundách určující interval rotování boxů. Hodnota **none** vypíná automatickou rotaci boxů. Defaultní hodnota při absenci atributu je **3000** (3 sekundy).
`speed`	*slow* | *fast* | *number*	Definuje rychlost animace boxu. Hodnota **number** určuje rychlost v milisekundách. Defaultní hodnota při absenci atributu je **slow**.
`slideNumber`	*number*	Definuje počet boxů, u kterých se v jednom okamžiku provede rotace. Defaultní hodnota při absenci atributu je **1**. Maximální hodnota atributu nesmí překročit celkový počet boxů v komponentě Slider.
`automaticWay`	*left* | *right*	Definuje směr rotace při automatické rotaci. Defaultní hodnota při absenci atributu je **right**.

6.3 Komponenta XtempFrame

Komponenta XtempFrame slouží pro demonstraci navržených komponent implementovaných nad preprocesorem XTemp. Při použití komponenty se vykreslí *textarea* s vloženým kódem spolu s boxem, kde se zobrazuje výsledek vykreslování. Obsah vstupního elementu *textarea* uživatel může jakkoliv měnit a vyzkoušet si tak funkčnost komponent. Kód umístěný v elementu *textarea* je vykreslován za použití technologií AJAX.

Obrázek 6.3: Komponenta XtempFrame.

6.3.1 Použití komponenty

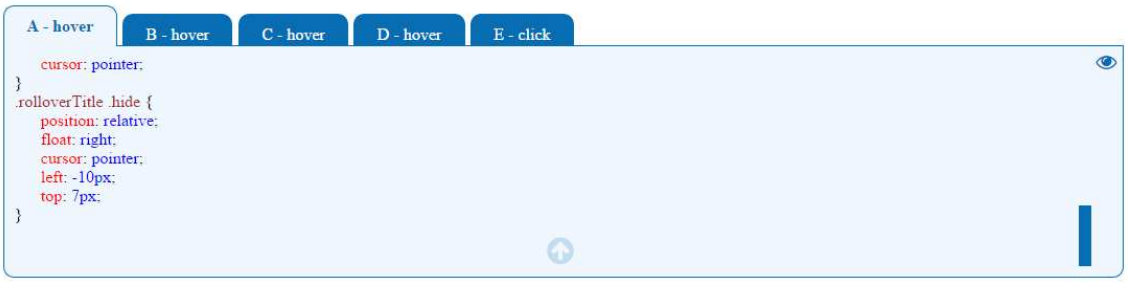
Pro použití komponenty slouží párový tag <ui:xtempFrame>, ohraničující kód k vykreslení. Tento párový tag je možné použít nad jakoukoliv ucelenou částí kódu dokumentu. Pro testování funkčnosti navržených komponent, jejichž kód je umístěn mezi tagy <ui:xtempFrame>, je potřeba deklarovat jejich prefix společně s jmenným prostorem, v němž jsou implementovány. Deklaraci je pro přehlednost ideální provést u párového elementu *html* ohraničujícího strukturu celého dokumentu. Při prvním vykreslení komponenty XtempFrame je element *html* spolu s deklarací jmenných prostorů extrahován a vykreslen do elementu *textarea*, tak aby obsah tohoto elementu reprezentoval ucelený HTML dokument k vykreslení. Pokud bychom po vykreslení XtempFrame komponenty chtěli otestovat funkčnost některé komponenty, jejíž jmenný prostor nebyl deklarován, je tuto deklaraci možné provést v samotném těle elementu *textarea*.

6.3.2 Popis implementace

Komponenta je implementována v souboru `XtempFrameElement.php`. Při prvním vykreslení je volána metoda `getChildIntoString()` třídy `XtempFrameElement`, kde se extrahuje kód ohraničený párovým tagem <ui:xtempFrame>. K extrahovanému kódu se přidá element *html* obsahující deklaraci jmenného prostoru a výsledný řetězec se použije jako defaultní hodnota elementu *textarea*. Poté se pomocí metody `process($src)` třídy `\XTemp\Filter`, kde se jako parametr *$src* použije extrahovaný kód, získá řetězec jenž představuje výsledek po zpracování Xtemp preprocesorem. Tento řetězec je vykreslen do xtempFrame boxu s výsledkem vykreslování. Při AJAXových požadavcích je metodě `process($src)` předána hodnota elementu *textarea*. Kompletní API komponenty XtempFrame je popsáno v příloze C.5 tohoto dokumentu.

6.4 Komponenta TabBox

Komponenta je určena pro strukturalizaci HTML obsahu. HTML obsah člení do boxů, které jsou omezeny na uživatelem předem danou velikost. V každém boxu je při defaultním nastavení implementované tzv. *goUP* tlačítko, pro skrolování na začátek komponenty TabBox a *showAll* tlačítko, pro zobrazení celého obsahu. Jednotlivé boxy s definovaným obsahem je při použití komponenty nutné uzavřít párovým tagem <ui:tabBox> . Všechny boxy je taktéž potřeba umístit do tzv. *tabContaineru*, jehož obsah je ohraničen párovým tagem <ui:tabContainer> .



Obrázek 6.4: Komponenta TabBox.

Tabulka 6.4: Atributy pro <tabBox>

Atribut	Hodnota	Popis
`title`	*string*	Nadpis tabBoxu.
`speed`	*slow* | *fast* | *number*	Definuje rychlost animace boxu. Hodnota **number** určuje rychlost v milisekundách. Defaultní hodnota při absenci atributu je **slow**.
`event`	*click* | *hover*	Definuje jaká událost zobrazí obsah konkrétního boxu komponenty TabBox. Defaultní hodnota při absenci atributu je **click**.
`showGoUP`	*yes* | *no*	Zobrazí *GoUP* tlačítko pro skrolování na začátek komponenty TabBox. Defaultní hodnota při absenci atributu je **yes**.

Tabulka 6.5: Atributy pro <tabContainer>

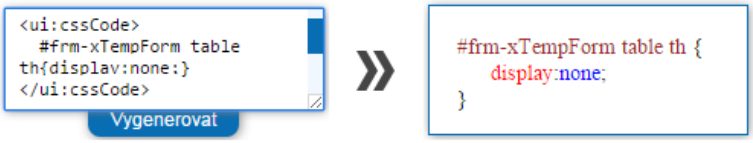
Atribut	Hodnota	Popis
loading	*number*	Určuje velikost containeru pro vnořené boxy. Při absenci tohoto atributu je velikost defaultně nastavena na velikost **500** pixelů.
loadPosition	*left* | *right* | *down* | *top*	Definuje na kterou stranu budou načteny "aktivní boxy s nadpisy" vůči obsahu komponenty TabBox. Defaultní hodnota při absenci atributu je **top**.

6.4.1 Popis implementace

Komponenta TabBox je implementována v souborech `TabContainerElement.php` a `TabBoxElement.php`. Soubor `TabContainerElement.php` popisuje implementaci elementu <tabContainer> , který plní funkci obalu jednotlivých boxů komponenty TabBox. Element <tabContainer> umožňuje použití atributů *loading* (pro určení velikosti při načtení) a *loadPosition* (pro definici strany, kde budou načteny aktivní boxy s nadpisy). Funkcionalita těchto atributů je závislá na javascriptové knihovně JQuery, která na základě hodnot zmíněných atributů upraví CSS. Stejným způsobem jsou implementovány i atributy elementu <tabBox> jehož implementaci popisuje soubor `TabBoxElement.php`. Každému *tabConteineru* i *tabBoxu* je generováno jedinečné ID, které k jednotlivým elementům umožňuje přistupovat jako k sobě nezávislým prvkům. Kompletní API komponenty TabBox je popsáno v příloze C.3 tohoto dokumentu.

6.5 Komponenta CssCode

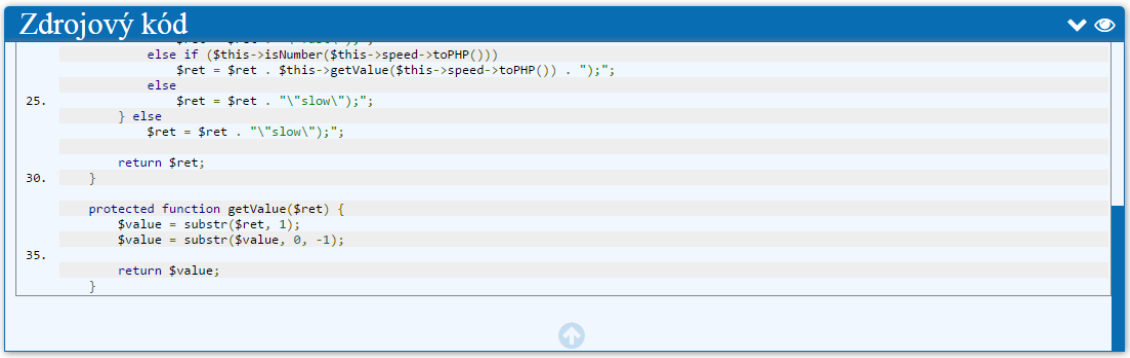
Komponenta cssCode slouží pro formátování a barevného rozlišení CSS. Stylopis CSS určen k formátování se při použití této komponenty obalí párovým tagem <ui:cssCode>. Barevné rozlišení je provedeno na základě jednoduchého principu selekce částí stylopisu CSS. Komponenta vyžaduje korektně definované CSS bez komentářů, proto se nehodí pro použití do praxe. Komponenta byla vytvořena pouze pro potřeby výukového webového tutoriálu popsaného v kapitole 7.



Obrázek 6.5: Komponenta CssCode.

6.6 Komponenta RolloverBox

Komponenta RolloverBox je stejně jako komponenta TabBox určena pro strukturalizaci HTML obsahu. Pro použití komponenty slouží párový tag <ui:rolloverBox> , který vymezuje HTML obsah k vykreslení.



Obrázek 6.6: Komponenta RolloverBox.

Tabulka 6.6: Atributy pro <`rolloverBox`>

Atribut	**Hodnota**	**Popis**
`title`	*string*	Nadpis komponenty RolloverBox.
`speed`	*slow* | *fast* | *number*	Definuje rychlost animace boxu. Hodnota **number** určuje rychlost v milisekundách. Defaultní hodnota při absenci atributu je **slow**.
`loading`	*hide* | *auto* | *number*	Definuje v jaké podobě bude rolloverBox zobrazen při načtení stránky. Hodnota `number` určuje velikost v pixelech. Hodnota **auto** definuje načtení v plné velikosti (podle obsahu rollBoxu). Hodnota **hide** při načtení zobrazí pouze nadpis *title*. Defaultní hodnota při absenci atributu je **hide**.
`showGoUP`	*yes* | *no*	Zobrazí *GoUP* tlačítko pro skrolování na začátek komponenty RolloverBox. Defaultní hodnota při absenci atributu je **yes**.

6.6.1 Popis implementace

Funkcionalita komponenty RolloverBox, implementovaná v souboru `RolloverBoxElement.php`, je velice podobná komponentě TabBox a její implementace stejně jako u této komponenty z velké části závisí na javascriptové knihovně JQuery. Každé komponentě je v podobě atributu přiřazen jedinečný identifikátor ID, který mimo jiné umožňuje neomezené zanořování jednotlivých *rolloverBoxů* do sebe. Téměř veškerá funkcionalita je implementována ve funkci `render()`, kde se kromě HTML statických elementů sestavuje javascriptový kód. Kompletní API s popisem funkcí je popsáno v příloze C.4.

Kapitola 7

Aplikace pro demonstraci funkčnosti navržených komponent

K demonstraci funkčnosti jednotlivých komponent byl vytvořen webový výukový tutoriál. V tomto tutoriálu jsou popsány všechny implementované komponenty. Ke každé komponentě je navíc přidána komponenta XtempFrame, pomocí které se může otestovat funkčnost, případně modifikace dané komponenty. Výukový tutoriál je vytvořen responzivní technologií, jeho použití tedy nelimituje velikost zobrazovacího zařízení. Díky tomuto faktu se dá snadno testovat i responzivnost jednotlivých komponent.

Obrázek 7.1: Podoba aplikace pro zařízení s minimální šířkou zobrazovací plochy 800px.

Obrázek 7.2: Podoba aplikace pro mobilní zařízení.

Kapitola 8

Testování

Testování výstupu jednotlivých komponent bylo prováděno komponentou *XtempFrame* popsanou v kapitole 6.3. Z důvodu převážně čistě grafického výstupu implementovaných komponent nebylo zapotřebí vytvářet sady automatických testů. I když k tomuto účelu Nette framework nabízí pohodlný nástroj *Nette Tester*, popsaný v kapitole 4.3. K testování tedy bohatě postačil spolu s komponentou *XtempFrame* ladící nástroj *Tracy* (popsaný v kapitole 4.3).

K ladění AJAXových operací byl použit ladicí nástroj *Firebug 2.0.9*, jehož autorem je Joe Hewitt. Tento nástroj představuje rozšíření prohlížeče *Firefox* a je šířen jako open-source projekt pod licencí BSD. Nástroj *Firebug* byl používán s rozšířením *FirePHP*, pro který má Nette framework zabudovanou implicitní podporu.

Pro ladění komponenty *XTempTable* (popsané v kapitole 6.1) byla použita data, která byla vygenerována online generátorem portálu *generatedata.com*. Jednotlivé výstupy seřazených dat byly porovnávány s výstupy vygenerovanými JQuery pluginem *tablesorter*[1], šířeným pod svobodnou licencí MIT, jehož autorem je Christian Bach.

Testování komponent proběhlo ve webových prohlížečích *Google Chrome* verze *42*, *Mozilla Firefox* verze *37*, *Opera* verze *27* a *Internet Explorer* verze *11*.

[1] Dostupným online na adrese *http://tablesorter.com/*.

Kapitola 9

Závěr

Cílem této bakalářské práce bylo navrhnout a implementovat komponenty uživatelského rozhraní s podporou AJAX. Komponenty měly být navrženy s možností vstupu i výstupu dat včetně grafického výstupu. Implementace komponent byla přizpůsobena jejich okamžitému nasazení do vyvíjené aplikace s minimálním zásahem do programového kódu. K tomuto účelu byly komponenty vytvářeny s volbou změny podoby a funkcionality přímo v XHTML souboru definujícího výslednou reprezentaci dat. Veškeré možnosti jednotlivých změn jsou iniciovány pomocí atributů, jejichž použití v komponentách je volitelné. Všechny atributy s jejich výčtem podporovaných hodnot včetně implicitní hodnoty použité při absenci daného atributu jsou zobrazeny u popisu jednotlivých komponent v této bakalářské práci. Úmysl okamžitého nasazení do vyvíjené aplikace navíc podporuje fakt, že jsou veškeré komponenty implementovány responzivním přístupem. Jejich použití v aplikacích tedy nevyžaduje přizpůsobení podoby pro zobrazovací zařízení s různou velikostí zobrazovací plochy. K testování a demonstraci funkčnosti jednotlivých komponent byl vytvořen webový vyukový tutoriál, kde jsou popsány všechny implementované komponenty.

Implementaci komponent předcházelo studium existujících jazyků a nástrojů pro tvorbu webové aplikace. Úkolem byl také výběr vhodného aplikačního rámce, v němž se měli navržené komponenty implementovat. Jako vhodný aplikační rámec byl doporučen český framework Nette. K výběru tohoto aplikačního rámce navíc přispěla předchozí zkušenost s prací s tímto frameworkem při vývoji webové aplikace jako školního projektu. Jako alternativy byly zvažovány frameworky Zend a CodeIngniter. Při jejich studiu byl největší prostor věnován jejich možnostem tvorby komponent a znovupoužitelných programových konstrukcí obecně. Spolu s frameworkem Nette byl pro implementaci komponent použit preprocesor XTemp, jehož základní popis s přínosem, který přináší jeho použití, je taktéž součástí této práce.

Používání komponent při vývoji webových aplikací vede k přehlednosti a strukturalizaci zdrojových kódů. Jejich používání, díky ucelené funkcionalitě, nebrání v mnoha rozdílných aplikacích, což vede k urychlení vývoje těchto aplikací. Rozšíření této bakalářské práce v rámci implementace dalších komponent může být nadsazeně řečeno neomezené. S tím ostatně počítá i preprocesor XTemp, jehož implementace je přizpůsobená k automatickému vykreslení nově vytvořené komponenty, bez nutnosti jakékoliv konfigurace. Příkladem rozšíření mohou být komponenty pro práci s databází, formuláři nebo komponenty pro filtrování či ověřování údajů. Zajímavým rozšířením mohou být i komplexnější komponenty představující například nákupní košík v internetovém obchodě, nebo mechanismus automatického zasílání elektronické pošty.

Literatura

[1] Asleson, R.; Schutta, N. T.: *Ajax: Vytváříme vysoce interaktivní webové aplikace.* Computer Press, a.s., 2006, 272 s., iSBN 80-251-1285-3.

[2] Beal, V.: API - application program interface[online]. Dostupné z: `http://www.webopedia.com/TERM/A/API.html`, [cit. 2015-25-04].

[3] Belchin, M.; Juberias, P.: *Web Programming with Dart.* Apress Berkely, CA, 2015, 472 s., iSBN 978-1-484205-57-0.

[4] Berners-Lee, T.; Fielding, R.; Masinter, L.: Uniform Resource Identifiers (URI): Generic Syntax[online]. RFC 2396, Network Working Group, srpen 1998 [cit. 2015-20-04].

[5] Böhmer, M.: *Zend Framework: programujeme webové aplikace v PHP.* Computer Press, a.s., první vydání, 2010, 416 s., iSBN 978-80-251-2965-4.

[6] Braunstein, R.: *ActionScript 3.0 Bible.* Indianapolis: Wiley, druhé vydání, 2010, 1008 s., iSBN 978-0-470-52523-4.

[7] Chaffer, J.; Swedberg, K.; Resig, J.: *Learning jQuery create better interaction, design, and Web development with simple JavaScript techniques.* Packt Publishing Ltd., Čtvrté vydání, 2013, 444 s., iSBN 978-1-78216-314-5.

[8] Drumelis, V.: 20 Best PHP Frameworks for Developers in 2014[online]. Dostupné z: `http://www.codegeekz.com/20-best-php-frameworks-developers-august-2014/`, 2014-19-08 [cit. 2015-22-04].

[9] ECMA-404: The JSON Data Interchange Format[online]. Technická zpráva, Ecma International, East Lansing, Michigan, říjen 2013 [cit. 2015-24-04].

[10] E.Holzschlag, M.: *HTML a CSS. jdi do toho.* Grada Publishing, a.s., první vydání, 2006, 105–107 s., iSBN 80-247-1454-X.

[11] Fielding, R. T.: *Architectural styles and the design of network-based software architectures.* Dizertační práce, University of California, 2000, iSBN 0-599-87118-0.

[12] Garrett, J. J.: Ajax: A New Approach to Web Applications[online]. Dostupné z: `http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications`, 2005-18-02 [cit. 2015-16-04].

[13] Grant, A.: *Beginning AngularJS.* Apress Berkely, CA, 2014, 200 s., iSBN 14-842-0161-2.

[14] Hanák, D.: Dependency injection - předávání závislostí[online]. Dostupné z: http://www.itnetwork.cz/dependency-injection-navrhovy-vzor, 2012 [cit. 2015-26-04].

[15] Howard, D.; Laurent, S. S.; Blanchette, M.; aj.: *Node.js for PHP Developers*. O?Reilly Media, Inc., první vydání, 2012, 269 s., iSBN 978-1-449-33360-7.

[16] James, L.; Ware, B.: *Open Source - vývoj webových aplikací: Linux, Apache, MySQL, Perl a PHP*. Praha: Mobil Media, první vydání, 2003, 448 s., iSBN 80-86593-43-6.

[17] Kosek, J.: *PHP - tvorba interaktivních internetových aplikací*. Grada Publishing, spol. s.r.o., první vydání, 1999, 492 s., iSBN 80-7169-373-1.

[18] Škultéty, R.: *Javascript. Programujeme internetové aplikace*. Computer Press, a.s., druhé vydání, 2004, 13–17 s., iSBN 80-251-0144-4.

[19] Lavin, J.: *AngularJS Services*. Packt Publishing Ltd., 2014, 152 s., iSBN 17-839-8356-6.

[20] MacDonald, M.: *Creating a Website: The Missing Manual*. O'Reilly Media, třetí vydání, 2011, 584 s., iSBN 978-1-4493-0172-9.

[21] Mačok, M.; Strádal, V.: NESMRTELNÝ CROSS-SITE SCRIPTING[online]. Dostupné z: http://doc.nette.org/cs/2.3/glossary/, březen 2005 [cit. 2015-22-04].

[22] Mardan, A.: *Practical Node.js*. Apress, 2014, 300 s., iSBN 978-1-4302-6595-5.

[23] Nette.org: Slovníček pojmů[online]. Dostupné z: http://doc.nette.org/cs/2.3/glossary/, 2015-28-01 [cit. 2015-22-04].

[24] O'Dell, J.: Node.js creator Ryan Dahl steps away from Node?s day-to-day[online]. Dostupné z: http://www.venturebeat.com/2012/01/30/dahl-out-mike-drop/, 2012-30-01 [cit. 2015-26-04].

[25] Schafer, S. M.: *HTML, XHTML a CSS. Bible pro tvorbu WWW stránek*. Grada Publishing, a.s., Čtvrté vydání, 2009, iSBN 978-80-247-2850-6.

[26] Skvorc, B.: Best PHP Framework for 2015 . SitePoint Survey Results[online]. Dostupné z: http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/, 2015-28-03 [cit. 2015-22-04].

[27] W3Techs: Usage of client-side programming languages for websites[online]. Dostupné z: http://w3techs.com/technologies/overview/client_side_language/all, 2015-16-04 [cit. 2015-16-04].

[28] W3Techs: Usage of JavaScript libraries for websites[online]. Dostupné z: http://w3techs.com/technologies/overview/javascript_library/all, 2015-16-04 [cit. 2015-16-04].

[29] W3Techs: Usage of server-side programming languages for websites[online]. Dostupné z: http://w3techs.com/technologies/overview/programming_language/all, 2015-19-04 [cit. 2015-19-04].

[30] World Wide Web Consortium: XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)[online]. Dostupné z: http://www.w3.org/TR/xhtml1/#xhtml, 2002 [cit. 2015-23-04].

[31] World Wide Web Consortium: W3C[online]. Dostupné z: http://www.w3c.cz/, [cit. 2015-19-04].

[32] Zend Technologies Ltd.: About[online]. Dostupné z: http://framework.zend.com/about/, [cit. 2015-26-04].

Příloha A

Obsah přiloženého CD

- `XTemp`
  - `readme.txt` - stručný popis XTemp preprocesoru + seznam způsobů jak otestovat implementované komponenty
  - `licence.txt` - licenční soubor
  - `XTemp_all` - všechny zdrojové soubory, včetně frameworku Nette a aplikace pro demonstraci funkčnosti navržených komponent, připravené k okamžitému použití
  - `manual.txt` - návod na zařazení XTemp preprocesoru do Nette frameworku
  - `xtempTable`[1] - adresář pro komponentu XtempTable
  - `slider`[1] - adresář pro komponentu Slider
  - `xtempFrame`[1] - adresář pro komponentu XtempFrame
  - `tabBox`[1] - adresář pro komponentu TabBox
  - `cssCode`[1] - adresář pro komponentu CssCode
  - `rolloverBox`[1] - adresář pro komponentu RolloverBox
  - `xTemp_API.pdf`[2] - API reference hlavních částí XTemp preprocesoru
- `bp_text`
  - `bp_tex` - zdrojové soubory bakalářské práce pro vysázení textu systémem LaTeX
  - `bp_komponenty.pdf` - bakalářská práce ve formátu pdf

[1] Adresáře jednotlivých komponent obsahují zdrojové soubory (včetně CSS), stručný popis komponenty a návod jak lze komponentu použít.

[2] API je totožné s přílohou D této bakalářské práce.

Příloha B

Ukázka zdrojového kódu

Zdrojový kód popisuje část implementace komponenty Slider, konkrétně rotaci boxů na pravou stranu.

```
/*$ret = javascriptovy retezec k~vykresleni
  $slideNumber = pocet boxu u~kterych se v~jednom okamziku provede rotace
  $next_cond = retezec posloupnosti jquery funkci .next() vygenerovany
               podle hodnoty promenne $slideNumber snizene o~1
  $next = retezec posloupnosti jquery funkci .next() vygenerovany podle
          hodnoty promenne $slideNumber
  $speed = rychlost animace boxu
*/
$ret .=  "$('#slider_right').click(rightRotate);

function rightRotate(){
  var child = $('#xtemp_slider .slider_box:first');";
  $next_for = "";
  //ulozi absolutni pozici vsech boxu, ktere budou rotovat
  for ($i=0;$i < $slideNumber;$i++){
    $ret .=
    "var first_left" . $i . "=child" . $next_for . ".position();";
    $next_for .= ".next().next()";
  }
  //posunuje pozice boxu o~$slideNumber dokud nenarazi na posledni box
  $ret .= "while (!child" . $next_cond . ".is(
        '#xtemp_slider .slider_box:last')){

    var position = child" . $next . ".position();
    child.animate({left:position.left}," . $speed . ");
    child = child.next().next();
  }";
  //provede rotaci poslednich $slideNumber boxu na pocatecni pozici
  for ($i=$slideNumber-1;$i >= 0;$i--){
    $ret .=
      "$('#xtemp_slider .slider_box:last').animate(
        {left:first_left" . $i . ".left}," . $speed . ");
      var prev_script = $('#xtemp_slider .slider_box:last').prev();
      $('#xtemp_slider .slider_box:last').insertAfter('#slider_right');
      prev_script.insertAfter('#slider_right');";
  }
 $ret .= "}";
```

Příloha C

Popis API navržených komponent

C.1 XTempTable API reference

`class XtempTableElement extends \XTemp\Tree\Element`

public	`loadParams()`	- Načtení nastavených atributů.
public	`beforeRender()`	-Zpracování jednotlivých tagů definovaných v XHTML souboru. - Vytvoření polí pro vykreslení záznamů tabulky. - Uložení záznamů ve formátu JSON do souboru json_bodyRows.txt.
public	`getSearchItems()`	- Vykreslení aktivních prvků pro víceúrovňové vyhledávání.
public	`getPaginator()`	- Dynamické vykreslování aktivních prvků určených pro ovládání stránkování.
public	`getValue()`	- Extrahování hodnot atributů pro následné zpracování.
public	`getValueOfShowParam()`	- Extrahování hodnot atributů určených pro rozhodnutí o vykreslení aktivních prvků.
public	`setParams()`	- Zpracování jednotlivých atributů.
public	`getInfoText()`	- Vykreslení informačního boxu.
public	`getNumberOfRows()`	- Vykreslení selektoru s možnostmi pro výběr počtu zobrazených záznamů na jedné stránce.

public	`getHeadTable()`	- Opakované vykreslování hlavičky tabulky.
public	`getSortType()`	- Extrahování typu řazení jednotlivých sloupců.
public	`render()`	- Předání výsledného řetězce, který bude šablonovacím systémem latte vykreslen do výsledné HTML podoby.

`class XTempTableAjaxRender`

public	`getHeadTable()`	- Opakované vykreslování hlavičky tabulky.
public	`setFilteredIndicator()`	- Nastavení indikátoru oznamujícího, že data byla filtrována.
public	`isFiltered()`	- Vrácení indikátoru oznamujícího, jestli byla data filtrována.
public	`getNumberOfAllItems()`	- Vrácení počtu nalezených záznamů.
public	`getSortArray()`	- Seřazení záznamů podle zvoleného sloupce.
public	`setSortedRows()`	- Přiřazení <slide> záznamů k seřazeným záznamům.
public	`getFilteredSearchText()`	- Vyhledávání zvoleného řetězce. - Nastavení vizualizace vyhledaného řetězce. - Vrácení záznamů obsahují zvolený řetězec. - Uložení filtrovaných dat do souboru.
public	`getInformationText()`	- Vykreslení informačního boxu.
public	`getOtherPage()`	- Načtení záznamů ze souboru. - Vytvoření polí pro vykreslení záznamů tabulky. - Uložení filtrovaných dat do souboru. - Vytvoření obslužných javascriptových funkcí. - Předání výsledného řetězce, který bude AJAXově vykreslen v načtené stránce.

`class XTempPresenter extends XhtmlPresenter` preprocesoru XTemp

public	`handleOtherPage()`	- Zajišťuje přenos parametrů mezi šablonou definující zdrojový kód komponenty *XTempTable* a metodou `getOtherPage` popsanou výše.

C.2 XTempSlider API reference

`class SliderBoxElement extends \XTemp\Tree\Element`

public	`render()`	- Vykreslování jednotlivých boxů. - Nastavení javascriptové funkce pro výpočet velikosti jednotlivých boxů. - Nastaveni javascriptové funkce pro nastavení absolutního pozicování jednotlivých boxů. - Nastaveni javascriptové funkce pro změnu nastavení velikosti a absolutního pozicování jednotlivých boxů při změně velikosti zobrazovací plochy.

`class SliderElement extends \XTemp\Tree\Element`

protected	`loadParams()`	- Načtení nastavených atributů.
protected	`isNumber()`	- Testování atributu na numerickou hodnotu.
protected	`getValue()`	- Vrátí hodnotu atributu.
public	`render()`	- Nastavení javascriptových funkcí pro animaci boxů, směr rotace, počtu boxů k rotaci, nastavení počtu viditelných boxů, nastavení posunu rotace na "slider šipky", nastavení automatické rotace.

C.3 XTempTabBox API reference

`class TabContainerElement extends \XTemp\Tree\Element`

public	`loadParams()`	- Načtení nastavených atributů.
protected	`isNumber()`	- Testování atributu na numerickou hodnotu.
public	`render()`	- Nastavení javascriptových funkcí pro vykreslení pozice aktivních boxů, určení velikosti zobrazovací plochy tabBoxů.

`class TabBoxElement extends \XTemp\Tree\Element`

protected	`loadParams()`	- Načtení nastavených atributů.
protected	`isNumber()`	- Testování atributu na numerickou hodnotu.
protected	`addSpeed()`	- Definuje rychlost animace jednotlivých tabBoxů podle zvoleného atributu.
protected	`getValue()`	- Vrátí hodnotu atributu.
public	`render()`	- Nastavení javascriptových funkcí pro nastavení událostí pro animace tabBoxů, skrolovací funkce na "GoUP" tlačítko, funkce na "showAll" tlačítko. - Vykreslení jednotlivých tabBoxů.

C.4 XTempRolloverBox API reference

`class RolloverBoxElement extends \XTemp\Tree\Element`

protected	`loadParams()`	- Načtení nastavených atributů.
protected	`isNumber()`	- Testování atributu na numerickou hodnotu.
protected	`addSpeed()`	- Definuje rychlost animace jednotlivých tabBoxů podle zvoleného atributu.
protected	`getValue()`	- Vrátí hodnotu atributu.
public	`render()`	- Nastavení javascriptových funkcí pro nastavení velikosti při načtení rolloverBoxu, rychlosti animace, skrolovací funkce na "GoUP" tlačítko, funkce na "showAll" tlačítko. - Vykreslení rollover boxu spolu se všemi aktivními prvky.

C.5 XTempFrame API reference

`class XtempFrameElement extends \XTemp\Tree\Element`

public	`setDocument(\DOMDocument $document)`	- Vytvoření kopie DOMDocumentu.
protected	`getChildIntoString()`	- Extrahuje obsah tagu XTempFrame elementu. - Vyvolá znovunačtení obsahu XTempFrame elementu preprocesorem XTemp. - Vytvoří řetězec pro načtení do "input boxu".
public	`render()`	- Vykreslení komponenty XTempFrame.

public	`createComponentXTempForm()`	- Vytvoření formuláře komponenty *XTempFrame* s "input boxem" definujícím zdrojový kód k opětovnému vykreslení.
public	`XTempFormSucceeded()`	- Iniciuje generování zdrojového kódu pro šablonovací systém Latte (pomocí preprocesoru XTemp). - Vytvoří instanci třídy `\Nette\Templating\Template()` a na jejím základě získá HTML řetězec k vykreslení - detekuje AJAXový požadavek.
public	`setXtempCode()`	- Vrátí řetězec, který bude použit jako dafaultní hodnota pro "input box" komponenty.

Příloha D

Popis API XTemp preprocesoru

`class Loader extends \Latte\Loaders\FileLoader`

public	`getContent($file)`	- Přepsání metody `getContent($file)` třídy `Latte\Loaders\FileLoader` (přepsání metody zajistí, aby šablonovací systém Latte zpracovával řetězec vygenerovaný preprocesorem XTemp).
public	`isExpired($file, $time)`	- Zjištění platnosti dat uložených v cache úložišti.
public	`getDepsFile($deps)`	- Vrátí název souboru pro uložení dat určených ke cachování.
public	`loadDeps()`	- Načtení dat z cache úložiště.

`class XhtmlPresenter extends \Nette\Application\UI\Presenter`

public	`formatTemplateFiles()`	- Určení dohledaní šablon. Přepsáním metody `formatTemplateFiles` třídy `Nette\Application\UI\Presenter` se změní defaultní nastavení dohledání šablon. V tomto konkrétním případě nastavíme načítání šablon s příponou XHTML.

`class Filter`

public	`getFile()`	- Vrátí aktuálně zpracovávaný soubor.
public	`process($src, $file)`	- Vrátí řetězec, jako výsledek po zpracovávání dat XTemp preprocesorem. Z tohoto řetězce vygeneruje šablonovací systém Latte výsledný HTML výstup.
public	`buildTree($src, $file)`	- Vrátí strom instancí tříd. Strom instancí tříd je vygenerován metodou `buildSubTree`, která rekurzivním voláním zpracuje načtená data. Princip vytvoření stromu je popsán v kapitole 5.2.
public	`buildSubtree($tree, $node)`	- Generování stromu instancí tříd.
public	`restructureTree($root)`	- Restrukturalizace stromu instancí tříd.
public	`prepareRendering($root)`	- Volání metod `beforeRender()` tříd implementujících konkrétní komponenty. Metody `beforeRender()` předpřipraví data, která jsou potřebná před vykreslením komponenty.
public	`getDependencies()`	- Vrátí závislosti.
public	`createComponent($element)`	- Spolu s metodou `create` třídy `TagLib` vytváří instance tříd jednotlivých komponent.
public	`loadXML($inputXML)`	- Vytvoření \DOMDocument objektu ze vstupního XML souboru.
public	`buildNamespaceTable()`	- Vytvoření tabulky, která ve formě položek pole představuje kompletní seznam inplementovaných *Namespace* komponent.

`abstract class Component`

public	`toString()`	- Vrátí řetězec reprezentující název dané třídy.
public	`setTree($tree)`	- Nastaví strom instancí tříd.
public	`getTree()`	- Vrátí strom instancí tříd.
public	`getResources()`	- Vrátí pole zdrojů.
public	`addResource()`	- Přidá položku do pole zdrojů.
public	`getParent()`	- Vrátí instanci třídy implementující "rodiče" (v hierarchii posloupnosti stromu instancí tříd) aktuálně zpracovávaného prvku.
public	`getChildren()`	- Vrátí instanci třídy implementují "potomka" (v hierarchii posloupnosti stromu instancí tříd) aktuálně zpracovávaného prvku.
public	`getChildElements()`	- Vrátí instance tříd implementující všechny "potomky" (v hierarchii posloupnosti stromu instancí tříd) aktuálně zpracovávaného prvku.
public	`removeChild($child)`	- Odstranění položky z pole "potomků" (v hierarchii posloupnosti stromu instancí tříd) aktuálně zpracovávaného prvku. Pole "potomků" obsahuje instance tříd, implementujících vykreslení jednotlivých elementů XHTML souboru. Položka pole určená k odtranění je dána argumentem `$child`.
public	`removeAllChildren()`	- Odstranění všech položek z pole "potomků" (v hierarchii posloupnosti stromu instancí tříd) aktuálně zpracovávaného prvku. Pole "potomků" obsahuje instance tříd, implementujících vykreslení jednotlivých elementů XHTML souboru.

public	`addChild($child)`	- Přidání položky pole obsahující instanci třídy, implementující "potomka" (v hierarchii posloupnosti stromu instancí tříd) aktuálně zpracovávaného prvku.
public	`addAll($list)`	- Přidání všech položek pole daných argumentem $list do pole "potomků" (v hierarchii posloupnosti stromu instancí tříd) aktuálně zpracovávaného prvku.
protected	`recursiveSetTree($root, $tree)`	- Rekurzivní sestavení stromu instancí tříd.
protected	`renderChildren()`	- Vykreslení všech "potomků" (v hierarchii stromu instancí tříd) aktuálně zpracovávaného prvku.
protected	`renderIf($test, $value, $code)`	- Vrátí řídící konstrukci (ve formě řetězce) představující makro {if}.
protected	`renderNotIf($test, $value, $code)`	- Vrátí řídící konstrukci (ve formě řetězce) představující negaci makra {if}.
protected	`renderSelect($test, $variants, [$error])`	- Vrátí řídící konstrukci (ve formě řetězce) představující makro {if} {else}.

`abstract class TagLib`

public	`create(\DOMElement $element, Context $context, \DOMDocument $document)`	- Vytvoření komponenty. Metoda nejprve zjistí, zda existuje metoda create*Nazev_komponenty* implementující danou komponentu. Pokud existuje, tak ji volá. Pokud daná metoda není implementována, kontroluje existenci třídy *Nazev_komponenty*Element. V případě existence této třídy vytváří její instanci, v opačném případě je volána metoda třídy XHTML extends `\Xtemp\Taglib`, která element vykreslí v nezměněném tvaru.
public	`unknownElement(\DOMElement $element, Context $context)`	- Vykreslení komponenty v nezměněném tvaru.

`class ComponentTree`

public static	`setRoot()`	- Nastavení kořenového prvku DOMDocumentu.
public	`getRoot(\DOMElement $element, Context $context, \DOMDocument $document)`	- Vrátí kořenový prvek DOMDocumentu.
public	`getFile(\DOMElement $element, Context $context)`	- Vrátí aktuálně zpracovávaný soubor.
public	`addDependency()`	- Přidá položku do pole závislostí, pokud již pole danou položku neobsahuje.
public	`getDependencies($tree)`	- Vrátí pole závislostí.
public	`getAllResources()`	- Vrátí všechny zdroje.
public	`recursiveGetResources()`	- Vrátí zdroje rekurzivně.
public	`render()`	- Vykreslení komponenty.
public	`dumpTree()`	- Vykreslení stromu instancí tříd. (Metoda je určena pro vývojové účely).
public	`recursiveDump()`	- Rekurzivní vykreslování stromu instancí tříd. (Metoda je určena pro vývojové účely).

`class Content extends Component`

public	`toString()`	- Vrátí řetězec reprezentující název třídy s parametrem daným textovou hodnotou daného elementu.
public	`toStr()`	- Vrátí textovou hodnotu daného elementu.
public	`render()`	- Vykreslí textovou hodnotu daného elementu.

`abstract class Element extends Component`

public	`toString()`	- Vrátí řetězec reprezentující název třídy s parametrem daným textovou hodnotou daného elementu.
public	`getAttribute($name)`	- Vrátí atribut, určen parametrem $name z pole atributů daného elementu.
public	`getId()`	- Vrátí atribut "id" z pole atributů daného elementu.
public	`getElementName()`	- Vrátí název elementu, bez prefixu určujícího *namespace* v němž je implementován.
public	`getSimpleName()`	- Vrátí název elementu, bez prefixu určujícího *namespace* v němž je implementován.
public	`renderStartElement()`	- Vykreslí první (startující) tag daného elementu.
public	`renderEndElement()`	- Vykreslí uzavírací tag daného elementu.
public	`renderAttributes()`	- Vykreslí všechny atributy daného elementu.
public	`renderAttribute($name)`	- Vykreslí atribut, určen parametrem $name z pole atributů daného elementu.
public	`loadParams()`	- Načtení parametrů.
public	`loadAttributes()`	- Načtení atributů do pole.
public	`checkId()`	- Přiřazení atributu "id" danému elementu.
public	`checkIdConstant()`	- Kontrola hodnoty atributu "id" daného elementu.
public	`generateId()`	- Generování hodnoty atributu "id" pro daný element.
public	`requireAttrPlain($name, [$allowed] )`	- Vrátí požadovaný atribut $name. Pokud atribut $name neexistuje vypíše výjimku.
public	`useAttrPlain($name, $default, [$allowed])`	- Vrátí požadovaný atribut $name. Pokud atribut $name neexistuje, vrátí hodnotu danou parametrem $default.

public	`requireAttrExpr($name)`	- Vrátí instanci třídy `Expression`, na základě atributu $name. Pokud atribut $name neexistuje, vypíše výjimku.
public	`useAttrExpr($name, $default)`	- Vrátí instanci třídy `Expression`, na základě atributu $name. Pokud atribut $name neexistuje, vrátí instanci třídy `Expression`, na základě parametru $default.
public	`requireAttrNum($name)`	- Vrátí hodnotu atriburu $name. Pokud hodnota atributu $name není numerická, vypíše výjimku.
public	`loadExternalTemplate($file)`	- Načte externí šablonu.
public	`addResourceTemplate($file, $params)`	- Přidá zdrojovou šablonu do stromu instancí tříd.

`class Entity extends Component`

public	`render()`	- Vykreslí textovou hodnotu daného elementu.